Wearable Circadian Rhythm Monitor	North Carolina State University

TI Innovation Challenge 2015 Project Report

Team Leader:	Jonathan Howell, jrhowel2@ncsu.edu
Team Members:	David Le, dkle@ncsu.edu David Brown, dwbrown3@ncsu.edu
Advising Professor:	Dr. John Muth, muth@ncsu.edu
Video	Provide link to video that you've uploaded to www.ti.com/videos (ti.com/videos was not working when we submitted) http://youtu.be/iny3Fk3ennc
Texas Instruments Mentor (if applicable):	
Date:	5/20/2015

Qty.	**List all TI analog IC and TI processor part number and URL**	**1) Explain where it was used in the project? 2) What specific features or performance made this component well-suited to the design?**
1	CC2541 (http://www.ti.com/product/CC2541/description)	The Texas Instruments CC2541 Microcontroller was chosen for our application because the MCU features a Bluetooth Low-Energy transmitter/receiver SoC. Using the CC2541 was advantageous to us because it reduced the number of components needed for BLE connectivity (rather than adding an additional IC just for BLE communication).
1	TMP006 (http://www.ti.com/product/TMP006/description)	The Texas Instruments TMP006 thermocouple sensor was chosen for skin temperature measurement. The TMP006 operates at 3.3V and obtains skin temperature measurements by absorbing IR energy emitted from an object it is pointed at, subsequently reading the voltage change in its internal thermocouple array and outputting a temperature measurement. This was beneficial because it allowed for a non-invasive way to determine a user's skin temperature, and also had a small footprint on the PCB.
1	LM3658 (http://www.ti.com/product/LM3658/description)	The Texas Instruments LM3658 LiPo battery charging IC was chosen to allow a user of the device to charge their device battery via a Mini-USB connection while also allowing for the possible future addition of AC charging. The LM3658, when connected to a 3.7V LiPo battery, charges the battery at 4.2V at a rate of 100 or 500mA (charge rate pin selectable) and switches to maintenance mode upon sensing a full battery charge. The support circuitry also featured charging status LEDs indicating battery level.
1	TPS62203 (http://www.ti.com/product/tps62203)	The Texas Instruments TPS62203 DC-DC Step Down Buck Converter was chosen to convert the 3.7V LiPo battery output (or the LM3658 output if no battery was connected) to the 3.3V required by system components for operation. The device also provided up to 300mA to the system, which is necessary when all components are active at the same time. The TPS62203 was chosen instead of a Low Drop-Out Regulator in an attempt to reduce heat dissipation inside the enclosure while also improving performance.

Submit your TI Innovation Challenge project to http://tiic-na.hartehanks.com. Your team is encouraged to post your project as early as possible- Your submission will be kept offline until the contest has officially closed!

Instructions:

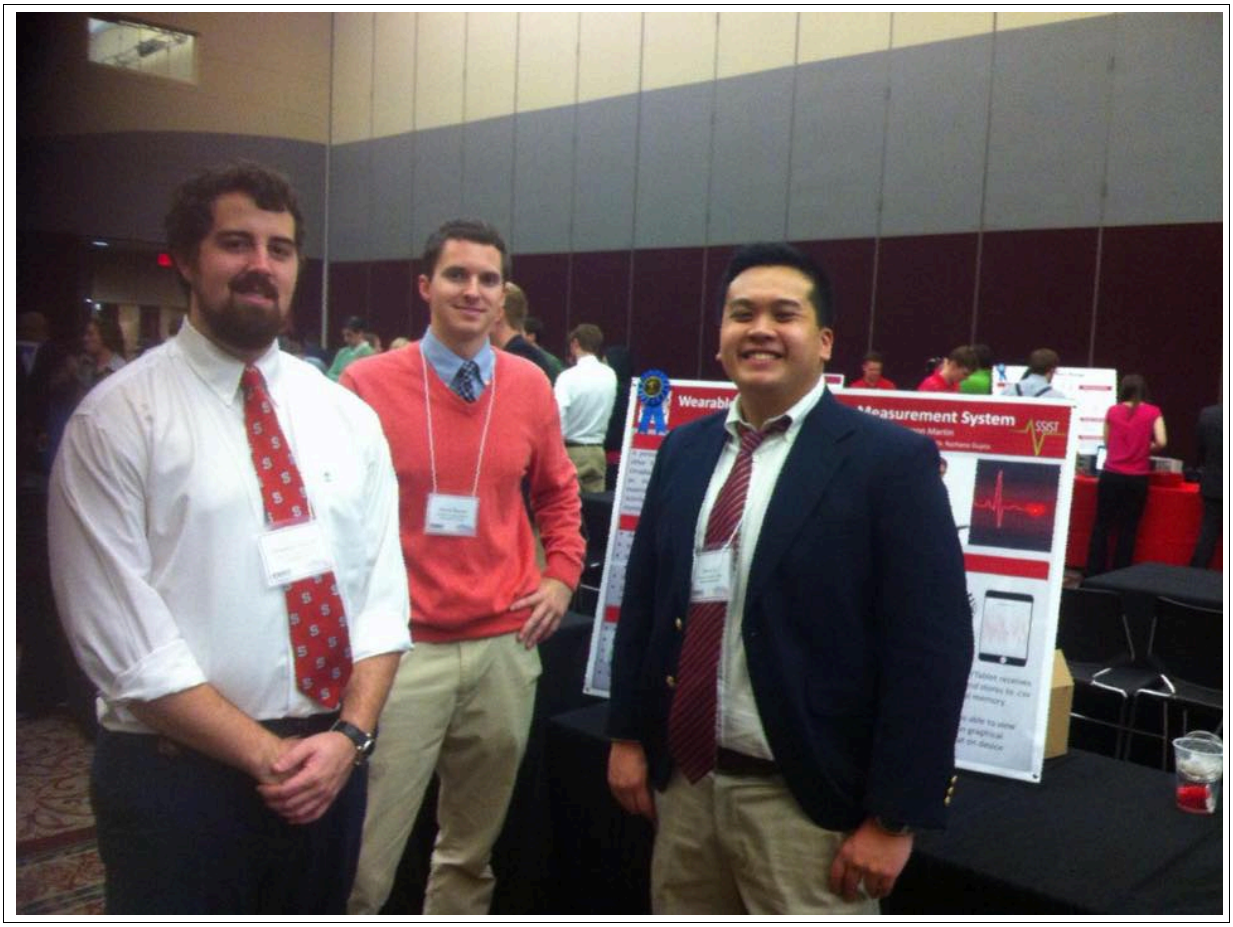
- Submit your project and include the following documents
  - Your full class report, which much include this TI project report.
  - Upload a video of demonstrating your project to www.ti.com/vidoes (must log into my.TI) and provide the link to that video in this project report. We'd love to see your team engaging with TI products!
  - Link to supplemental photos

Project abstract (a short high level written description of the design and motivation behind project), 1,000 words max:

Among the growing popularity and excitement surrounding the wearable fitness/sleep tracker market, a need for a cheaper and more accurate device has emerged. The purpose of this project was to produce a functional prototype which will effectively monitor an individual's circadian rhythm while also addressing existing accuracy and cost issues related to fitness/sleep tracking devices.

The function of our system is based on the acquisition and interpretation of data pertinent to a user's circadian rhythm. Using appropriate sensors, the system will acquire, process, and display measurements of a variety of physiological and environmental variables for a user of the device i.e. heart rate, skin temperature, motion, and ambient/UV light exposure. The sensor data is stored in the internal memory of the device and can be subsequently transmitted via Bluetooth Low-Energy to a PC or a smartphone running the Android OS. Once the device and data have been synchronized with and transmitted to an appropriate peripheral device, post-processing is performed and the data is displayed in either data vs. time formatting or as raw measurements in a .csv file. The system enables an individual to conveniently view data on their smartphone and continually log data in hopes of drawing conclusions about how they can improve the quality and regularity of their sleep and circadian rhythm.

Link to addition team photos: http://imgur.com/a/Mf9WY#0



Please submit your class report with this one page document. Your class report should include the following (Max of 10 pages, excluding appendix):

- Table of contents
- List of figures and tables
- A detailed written description of the project design
- Hardware Design
- Any Software Architecture used (include any software code as part of Appendix)
- Testing and Results / Conclusions
- Future Work / Recommendations
- Acknowledgements and/or References
- Appendix: schematics, CAD drawings, Critical IC Bill of Materials, User Manual, etc.

Wearable Circadian Rhythm Monitor

Final Design Document

Project Team
David Brown
Jonathan Howell
David Le

Project Sponsor
ASSIST Center

Project Advisors
Dr. John Muth
Dr. Rachana Gupta

Table of Contents

1. Project Description
2. System Architecture
3. Hardware Design
    3.1 Sensor Subsystem
    3.2 Power Management Subsystem
    3.3 Microcontroller, Memory, and Bluetooth Transmission Subsystem
4. Software Design
    4.1 MCU Firmware Design
    4.2 MCU Application Layer
    4.3 Android Application User Interface and Software Design
5. Mechanical Design
    5.1 Assembly
    5.2 Device User Interface
6. Testing, Results & Conclusions
7. Future Work & Recommendations

Appendix A - Schematic
Appendix B - Board Layout
Appendix C - Bill of Materials
Appendix D - Project Costs
Appendix E - User Manual
Appendix F - Bluetooth Low Energy UUID Table
Appendix G - Datasheets and References
Appendix H - CC2541 Firmware File Descriptions
Appendix I - Android Application File Descriptions
Appendix J - CC2541 Firmware Source Code
Appendix K - Android Application Source Code

Tables and Figures

Figure 1 - System Architecture
Figure 2 - Hardware Block Diagram
Figure 3 - Splash Screen and Device Selection Screen
Figure 4 - Syncing Screen and Real Time Screen
Figure 5 - Data Selection Screen and Data Graph Screen
Figure 6 - Enclosure Diagram
Figure 7 - 3D Printed Enclosure

1. Project Description

Among the growing popularity and excitement surrounding the wearable fitness/sleep tracker market, a need for a cheaper and more accurate device has emerged.

The purpose of this project was to produce a functional prototype which will effectively monitor an individual's circadian rhythm by examining their skin temperature, activity level, heart rate and light exposure. In addition to measuring these parameters, our project goal simultaneously addresses existing accuracy and cost issues related to fitness/sleep tracking devices.

The device implements readily available components and sensors in a wearable form-factor allowing the user to monitor their sleep habits and physiological metrics with ease and comfort. Additionally, the device employs a simple and clean user interface with a non-intrusive form factor resemblant of a wrist watch.

2. System Architecture

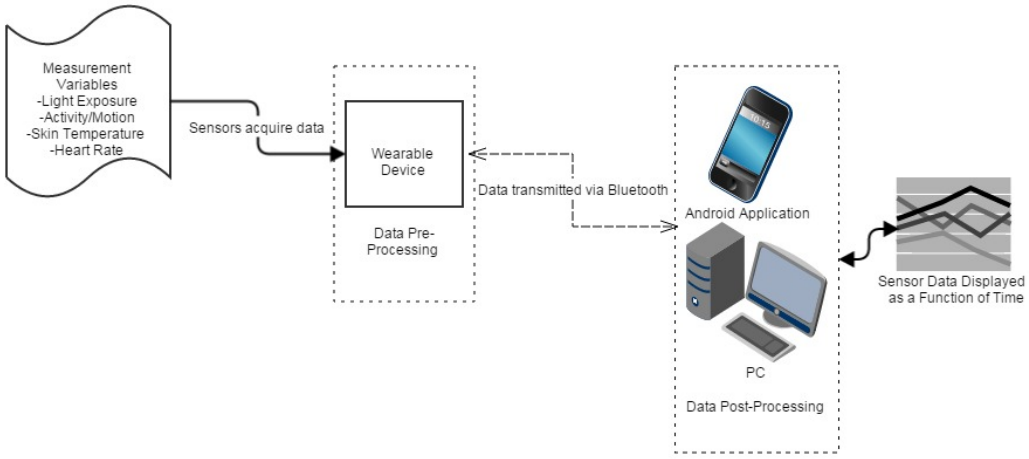


Figure 1 - System Architecture

The function of our system is based on the acquisition and interpretation of data pertinent to a user's circadian rhythm. Using appropriate sensors, the system will acquire, process, and display measurements of a variety of physiological and environmental variables for a user of the device i.e. heart rate, skin temperature, motion, and ambient/UV light exposure. The sensor data is stored in the internal memory of the device and can be subsequently transmitted via Bluetooth Low-Energy to a PC or a smartphone running the Android OS. Once the device and data have been synchronized with and transmitted to an appropriate peripheral device, post-processing is performed and the data is displayed in either data vs. time formatting or as raw measurements in a .csv file. The system enables an individual to conveniently view data on their smartphone and continually log data in hopes of drawing conclusions about how they can improve the quality and regularity of their sleep and circadian rhythm.

3. Hardware Design

The following diagram shows a very high level view of our system and each of its components. The individual components and the necessary supporting circuitry can be seen in Appendix A - Schematic. The PCB layout of our final iteration of our system can be seen in Appendix B - Board Layout. Additionally, the necessary Bill of Materials and the project cost can be seen in Appendix C - Bill of Materials and Appendix D - Project Costs.

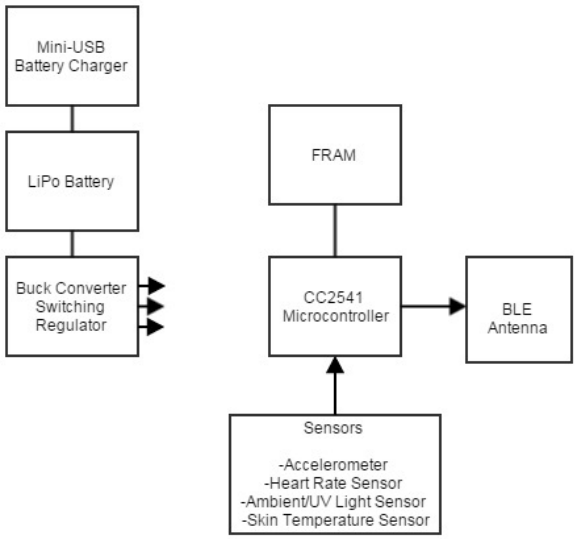


Figure 2 - Hardware Block Diagram

3.1 Sensor Subsystem

Accelerometer

The Analog Devices ADXL345 3-axis accelerometer was used as the motion sensor. The ADXL345 operates with a 3.3V supply voltage. The component was chosen for several reasons, including ultra-low power consumption, I^2C interfacing, and digital output data.

Heart Rate

The Silicon Labs Si1143 light sensor was chosen as the heart rate sensor. The Si1143 operates at 3.3V. Using the widely employed heart-rate measurement technique involving the sensing of light absorption by hemoglobin in blood (i.e. pulse oximetry), the Si1143's capabilities were perfect for our desired application. Using absorption measurements obtained from IR and Red (visible light)LEDs in conjunction with a software algorithm for processing the measurements, the sensor was capable of obtaining accurate heart rate figures. Additionally,The Si1143 was chosen for its I^2C interfacing and digital data output.

Ambient/UV Light

The Silicon Labs Si1132 light sensor was chosen as our ambient and UV light sensor. The Si1132 operates at 3.3V The Si1132 is capable of simultaneously acquiring ambient and UV light readings, making it a viable solution for our application. Additionally,The Si1132 was chosen for its I^2C interfacing, low power consumption, and digital data output.

Skin Temperature

The Texas Instruments TMP006 thermocouple sensor was chosen for skin temperature measurement. The TMP006 operates at 3.3V and obtains skin temperature measurements by absorbing IR energy emitted from an object it is pointed at, subsequently reading the voltage change in its internal thermocouple array and outputting a temperature measurement. Careful board layout considerations were taken into effect with the sensor in order to thermally isolate it from the rest of the circuit components, ensuring accurate data. The TMP006 interfaces via I^2C and outputs digital data.

3.2 Power Management Subsystem

Mini-USB Charging Circuitry and LiPo Battery

The Texas Instruments LM3658 LiPo battery charging IC was chosen to allow a user of the device to charge their device battery via a Mini-USB connection while also allowing for the possible future addition of AC charging. The LM3658, when connected to a 3.7V LiPo battery, charges the battery at 4.2V at a rate of 100 or 500mA (charge rate pin selectable) and switches to maintenance mode upon sensing a full battery charge. The support circuitry also featured charging status LEDs indicating battery level.

Buck Converter Switching Regulator

The Texas Instruments TPS62203 DC-DC Step Down Buck Converter was chosen to convert the 3.7V LiPo battery output(or the LM3658 output if no battery was connected) to the 3.3V required by system components for operation. The device also provided up to 300mA to the system which is necessary when all components are active at the same time. The TPS62203 was chosen instead of a Low Drop-Out Regulator in an attempt to reduce heat dissipation inside the enclosure while also improving performance.

3.3 Microcontroller, Memory, and Bluetooth Transmission Subsystem

The Texas Instruments CC2541 Microcontroller was chosen for our application because the MCU features a Bluetooth Low-Energy transmitter/receiver SoC (system-on-chip). Using the CC2541 was advantageous to us because it reduced the number of components needed for BLE connectivity (rather than adding an additional IC just for BLE communication). The BLE pins of the CC2541 only needed to be routed to a surface-mount BLE antenna in order for data transmission to be possible. Additionally, extra memory was required on top of the 256k memory of the CC2541 to store sensor data, so the FM24V10 FRAM memory IC was chosen. This provided ample storage capacity for our application and allowed data to be stored temporarily without the need for continuous Bluetooth communication..

4. Software Design

4.1 MCU Firmware Design

Our device's firmware uses Texas Instrument's open source SensorTag code as the basis for its design. The reason why we chose to use this open source provided by TI is because it helped simplify of all our event processing, sending, and communication (BLE) events for our system. Therefore during development of our device, we saved time by not having to worry about underlying event processing and message sending firmware design.

This allowed us to focus on developing firmware to enable the communication and use of our device sensors (SI1132, SI1143, ADXL345, TMP006) with the CC2541. Furthermore this allowed us to also focus on developing firmware for the CC2541 to use external FRAM (FM24V10) as well as developing software that controls sensor execution for data polling as well as a Unix Time Clock implementation to keep track of real time as our device does not have a RTC built into it.

4.2 MCU Application Layer

In the application layer of our device, we implemented the firmware which enabled the use of our sensors to communicate with the CC2541 using I2C protocol. The sensors that run on I2C protocol are the ADXL345 used for acceleration and movement measurements, the SI1132 used for visual, IR, and UV light measurements, the TMP006 used for ambient temperature measurements, and the SI1143 which is used for the heart rate monitoring by using an array of Red and IR LEDs with the optical sensor for pulse oximetry. In addition to these sensors,we implemented firmware for a FM24V10 FRAM IC which communicates to our MCU using I2C and provides our device the use of 128KB of external FRAM.

In regards to overall implementation and integration of our system using these sensors, software was written in the MCU application layer to initialize and poll our sensors in specified intervals to gather the sensor data and store it to FRAM.

Software code for polling our sensors was written with the limitation of our 128KB FRAM in mind. With 128KB of FRAM and all data being 11 bytes, it was decided that the accelerometer was polled every 1 minute, and all other sensors every 15 minutes. This gave us a total of 60 motion samples an hour, while giving us 4 samples of light, temperature, and heart rate every 15 minutes.

In addition to all the software and hardware integration code, we implemented a UTC clock which gets the current time and date upon phone/mcu synchronization and stores it into Unix Time Code format. This was done in order to compensate for not having a RTC implemented in our design.

4.3 Android Application User Interface and Software Design

Software was written for Android OS devices using java. The Android application serves as the primary user interface for interacting with their device and collected data. The application provides two modes of data collection/analysis; a real time mode and a synchronize-store more. The following diagrams demonstrates the use of the Android application.

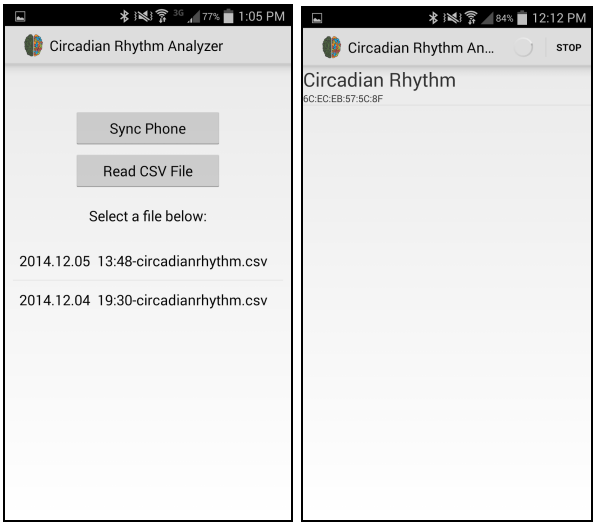


Figure 3 - Splash Screen and Device Selection Screen

The initial screen a user is presented with allows them to select between syncing the phone or viewing the collected data stored on the Android Device.

- “Sync Phone” allows the user to synchronize the device’s date and time settings to match the Android Device, synchronize the data from device to the Android device, or enable real time data viewing.
- “Read CSV File” allows the user to select an available CSV file stored on the Android device’s memory.

Upon selecting “Sync Phone” the user is presented with the next screen listing devices that implement Bluetooth Low Energy. A device listed as “Circadian Rhythm” should be selected. Each device shows it’s address below it allowing a user to differentiate between devices with the same or similar names. Selecting a “Circadian Rhythm” option takes the user to the next screen.

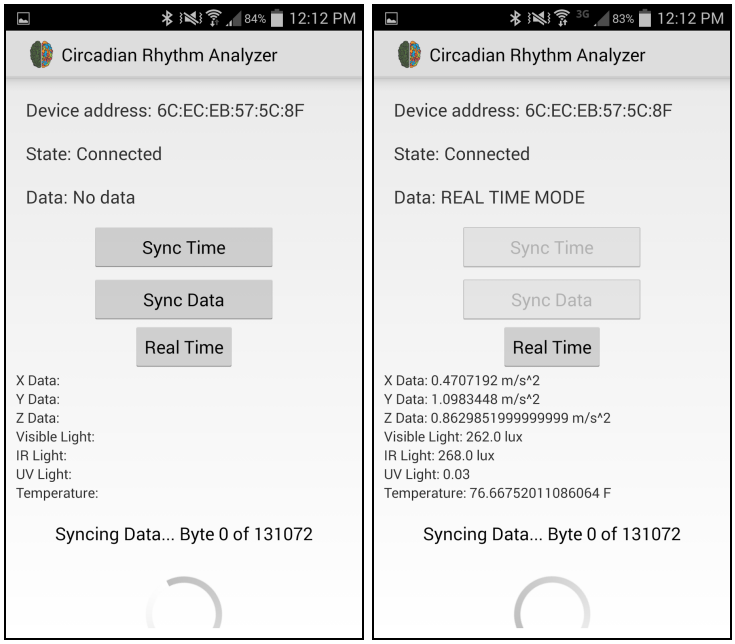


Figure 4 - Syncing Screen and Real Time Screen

The next screen allows the user to select between “Sync Time”, “Sync Data”, and “Real Time”. To activate enable data collection on the device, a user must select “Sync Time” (this must be performed every time power is cycled or the battery dies on the circadian rhythm device. “Sync Data” will provide syncing progress at the bottom of the screen. A user may not select to view real time data and synchronize data simultaneously. To select an option the user must return to the previous screen and select the appropriate device from the list of available devices again.

- “Sync Time” will synchronize the device’s date and time settings to match the Android device and also enable data collection on the circadian rhythm device.
- “Sync Data” will synchronize the data from device to the Android device.
- “Real Time” will enable real time data viewing.

Upon selecting “Real Time” the raw data collected from the circadian rhythm device is updated and showed on the bottom of the screen. This can be seen in the screenshot above.

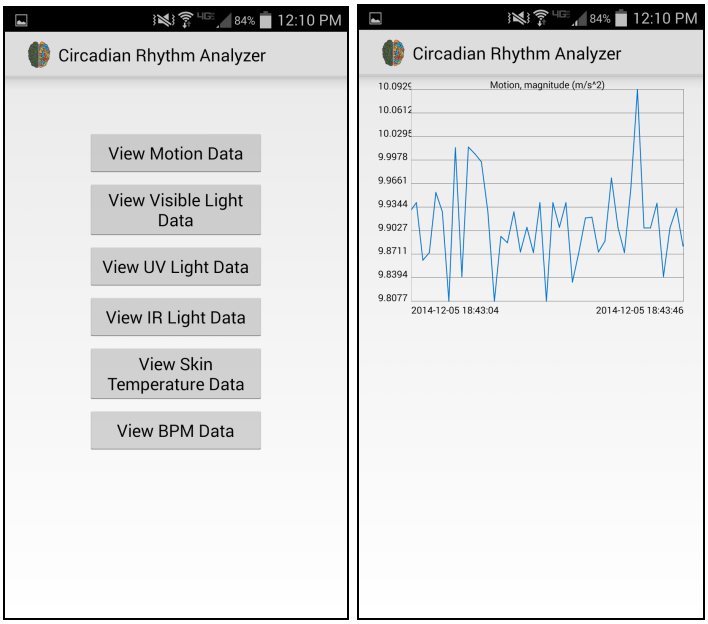


Figure 5 - Data Selection Screen and Data Graph Screen

Back at the initial screen where the user is presented with the option of “Sync Phone” and “Read CSV File”, upon selecting the latter a user is brought to the next screen allowing them to specify the type of data they want to have graphed.

Upon selecting one of the “View * Data” (where * is the type of data), a user is brought to the next screen where the data is graphed in a date-time vs. data-value, x vs. y format. The user can turn their android device horizontally to change screen orientation, pinch to zoom in and out, and scroll left and right to see the data-value as time changes.

5. Mechanical Design

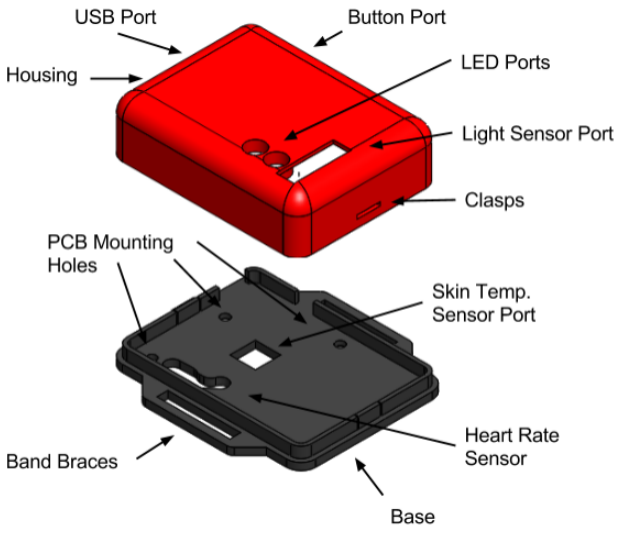


Figure 6 - Enclosure Diagram

Housing is a shell casing that flushly envelopes the PCB board and battery and secures it to the user's wrist for wearability. It consists of a base, cover and band.

5.1 Assembly

The PCB board is mounted to the base by way of 2-56 size screws that come up through the base at the mounting holes through 0.156" standoffs attached to the board through its own mounting holes. Once secured, the cover is then clasped down over the base and PCB board after threading the button and LEDs through their respective ports. The USB port should also be aligned with the USB opening. With these in place the band is then fed through it's braces and then wrapped around the user's wrist in such a way as to expose the user's skin to the opening in the base when worn.

5.2 Device User Interface

The device employs a simple user interface, allowing for easy and intuitive operation. A SPDT switch was used to toggle the enable pin of the Buck converter, allowing the user to power on the device by tieing the enable pin high, or powering off the device by tieing the enable pin to ground. A push-button switch connected to one of the GPIO pins of the microcontroller allows the user to initialize Bluetooth broadcasting with a single press of the button.

Two status LEDs provided power state and Bluetooth transmission indication. One LED remains on when the device is powered on, and the other blinks rapidly when the button is pressed and Bluetooth begins searching for connectivity.

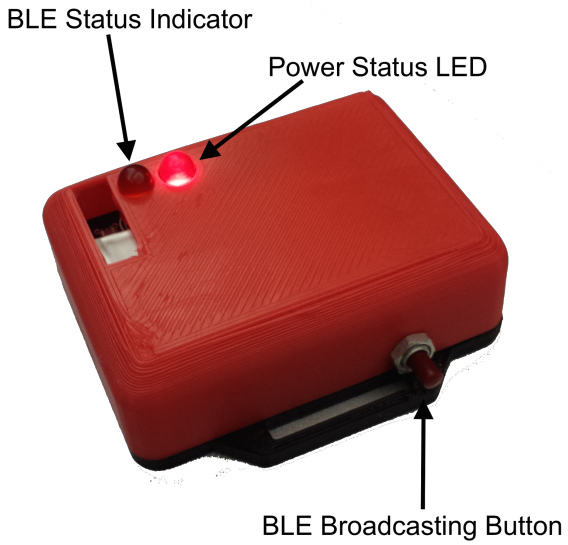


Figure 7 - 3D Printed Enclosure

6. Testing, Results & Conclusions

Upon testing our device, we were able to gather the all of the parameters related to a user's circadian rhythm that we planned for in our initial project goals. The degree of accuracy of the sensors varied, the light sensor and accelerometer providing very accurate measurements and the heart rate and skin temperature sensors lacking in accuracy. This lack of accuracy was likely due to the inadequate interfacing of the sensors to the parameter they were analyzing i.e. the surface of the user's skin. The accuracy was also impaired by the data outliers that occurred during sensor readings. This data was verified using existing products that are available on the market as well as tools such as an infrared heat gun and pulse oximeter. Despite the outliers, we concluded that the device could

realistically be used to gather usable data pertaining to a user's motion and light exposure, while also gathering reasonable data pertaining to an individual's heart rate and skin temperature. We also determined that modifications to the enclosure design would be necessary in order for the device to more accurately gather heart rate and skin temperature measurements. This was due to interference between the two sensors, between the sensors and the skin, and between the sensors and the enclosure.

7. Future Work & Recommendations

Future work that can be made to our system that would improve the overall performance is listed below.

- Heart Rate BPM Counter: Due to lack of knowledge on the subject of pulse oximetry, the algorithm created for the BPM counter was made off an interpretation of a research paper on the subject. Further exploration and research on acquiring a BPM count is needed.

- Improved Synergy: The PCB could be redesigned along with the mechanical enclosure to provide a better relationship between the two. This would allow for the PCB to be moved closer to the skin and would likely improve skin temperature and heart rate measurements. This would also allow for changes to be made to the enclosure that would decrease interference between sensors that rely on similar resources (such as infrared light).

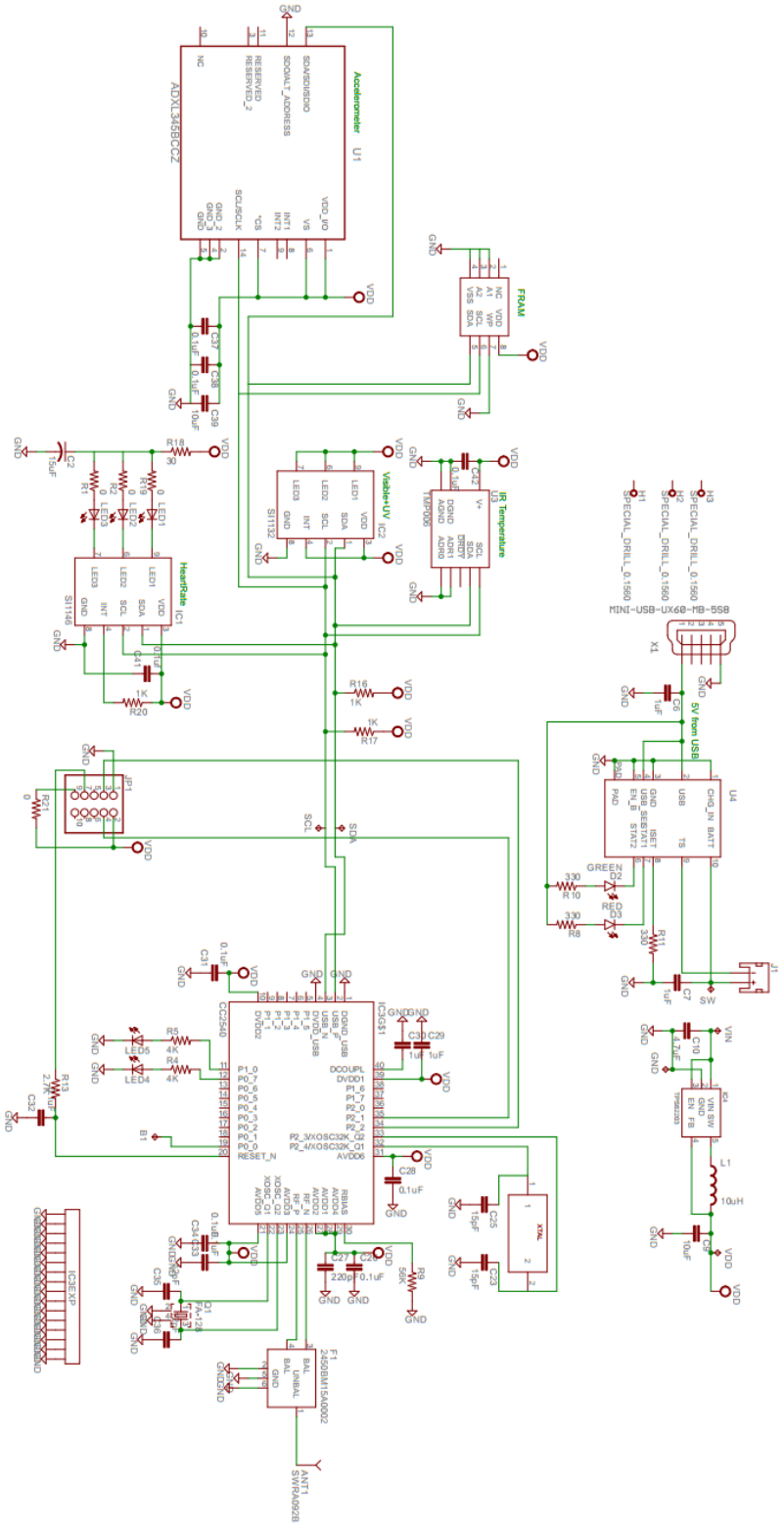
Additionally, there is some room for system improvements beyond our original scope of work. These potential improvements and features are listed below.

- Improved Device User Interface: Rather than using an LED based interface the device could incorporate a low power screen alternative such as an OLED or E-Ink display.

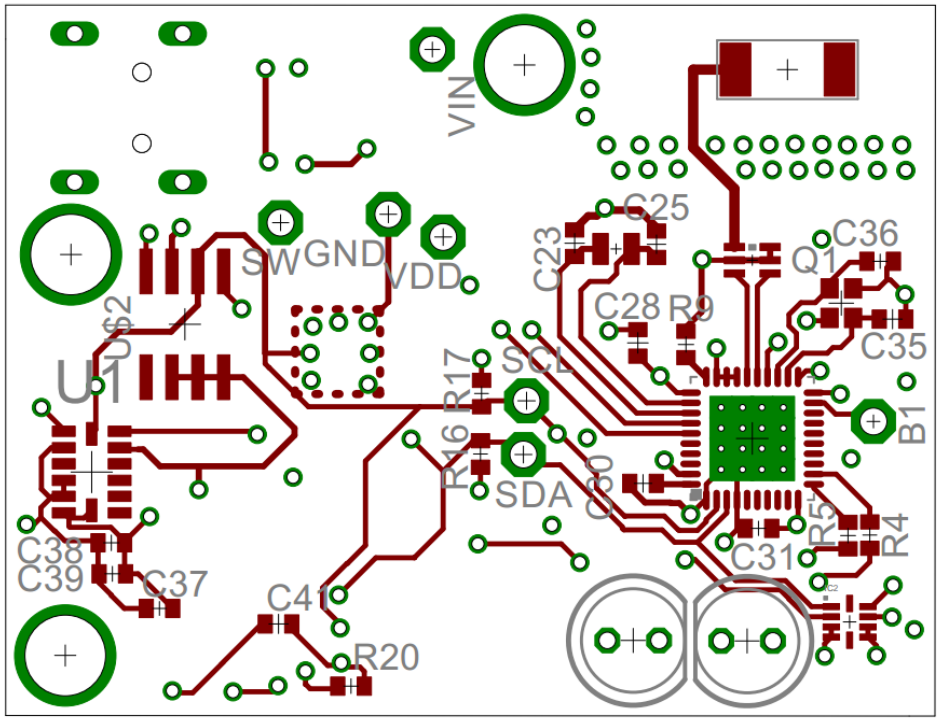
- Galvanic Skin Response: Another potentially important metric that could be measured if GSR was incorporated into the system. Knowing an individual's GSR could help reduce errors during sensor readings. That data may also be useful when put alongside data from the other sensors. This could be done via a Wheatstone bridge op amp circuit connected to the microcontroller's ADC.

- Data Analysis: This project focused primarily on gathering and presenting the collected data in a way that would be meaningful for a professional to examine and determine trends. It would be ideal if more data analysis could be done that would aid the professionals and save them time in processing the data. For example, if certain events were to happen then a flag could be raised, alerting the professional to a disturbance in an individual's circadian rhythm.

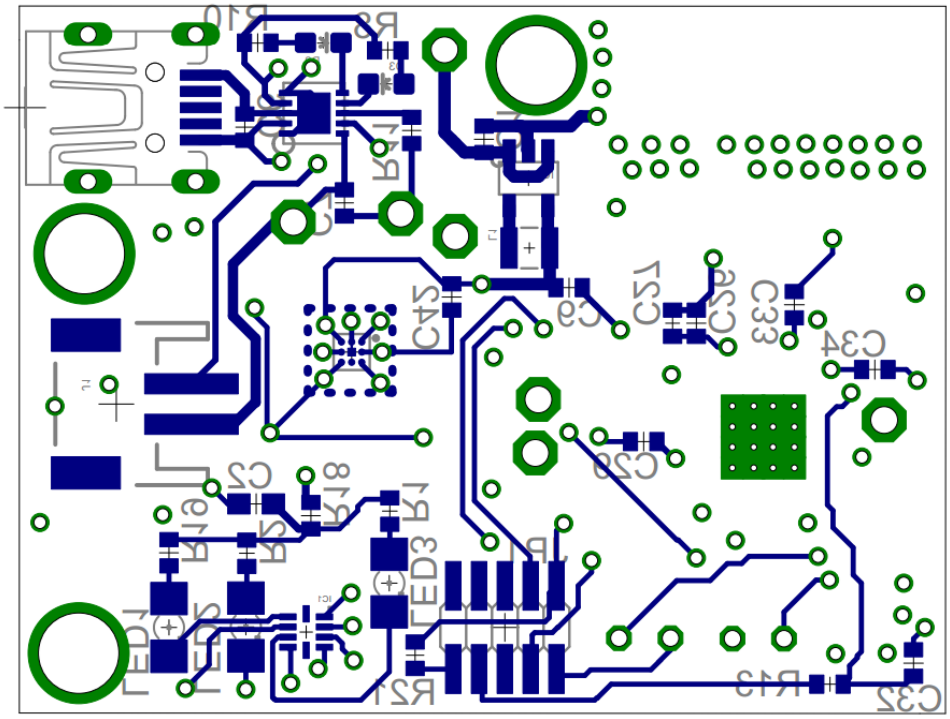
Appendix A - Schematic



Appendix B - Board Layout



Top



Bottom

Appendix C - Bill of Materials

Designator	Quantity Per Board	Value/Description
C7	1	1uF
C10	1	4.7uF
C2 (SI1143)	1	15uF
C6	1	1uF
C39	1	10uF
C9	1	10uF
C23, C25	2	15pF
C26, C28, C31, C32, C33, C34, C37, C38, C41, C42	10	0.1uF
C27	1	220pF
C29, C30	2	1uF
C35, C36	2	12pF
L1	1	10uH
Antenna	1	SMD BLE Antenna Chip
F1	1	Balun
IC1	1	SI1143
IC2	1	SI1132
IC3	1	CC2541 - 256k
IC4	1	TPS62203
U2	1	FM24V10 - FRAM
U1	1	ADXL345BCCZ
U3	1	TMP006
U4	1	LM3658
Mini USB	1	UX60-MB-5s8
J1	1	JST_2mm_male
JP1	1	CC debugger header
X1	1	FA-20H (32MHz)
X2	1	32.768KHz
D2, D3 (power supply indicators)	2	Red LED, Green LED
LED4, LED5	2	Red MCU LEDs
2 IR LEDS (Heart Rate)	2	HR IR LED
Other Red LED (heart rate)	1	HR RED LED
Low Debounce Button	1	-

SPDT Switch	1	-
Plastic Standoffs	3	0.156" threaded
Metal Standoffs	3	0.156" threaded
R1, R2, R19, R21	4	0
R8, R10, R11	2	330
R9	1	56K
R4, R5	2	750
R13	1	2.7K
R16, R17, R20	3	1K
R18	1	30
3.7V LiPo Battery	1	-

Appendix D - Project Costs

Prototype Costs

Discrete Components (Resistors, Caps, Inductors, LEDs, Crystals) - $25
ICs (Sensors, Memory, Microcontroller, Power Management) - $40
Battery - $5
Mechanicals (Mounting, Connectors, 3D Enclosure Print, PCB) - $15
Total Costs for Single Device - $85 (when ordering minimum quantity of parts)

Development Costs

Our development costs consisted mostly of eval boards for system components, two PCB revisions, and the components required to populate those PCBs.

Evaluation Boards/Development Tools Cost - $218

- MSP430G2 LaunchPad ~ $12 x 2
- CC2541 SensorTag Development Kit - $26
- CC Debugger - $50
- TMP006 Evaluation Kit - $25
- SI114x Evaluation Kit - $60

Rev A PCB Boards x 6 - $75
Rev B PCB Boards x 3 - $24
All Components for PCBs - $300
Hunt Library 3D Printing Services - $40
Total Development Costs - $624

Appendix E - User Manual

Introduction

Congratulations on being the proud owner of the Wearable Circadian Rhythm Measurement System! The Circadian Rhythm Monitor is a wristband worn device designed to track the parameters used to determine an individual's circadian rhythm. By wearing this device and examining the data it collects you will be able to determine your own personal circadian rhythm cycles and determine what your rhythms are most affected by. This device was developed with medical applications in mind and is just as likely to have been referred to you by your treating physician. If this is the case please remember to share your collected data with your physician.

What Are Circadian Rhythms?

Everybody is familiar with the times of day when they are most wakeful or sleepy. Circadian Rhythms are the cycles by which bodily systems synchronize with a day to day cycle. They are not the same as, but are driven by a your biological clock.

How Do They Affect Me?

Circadian Rhythms influence an individual's sleep-wake cycles, hormone release, body temperature and other bodily functions. For example, a disturbance in your general daily rhythms have been linked to various sleep disorders such as insomnia and "jet-lag." In some extreme cases, abnormal circadian rhythms have been associated with obesity, diabetes, depression, bipolar disorder and seasonal affective disorder.

How Will My Monitor Help Me?

Circadian Rhythms are affected by a number of signals from one's environment, the most significant being light exposure. In order to track your own circadian rhythms this device has been outfitted with light sensors capable of tracking UV, IR, and visible light intake. The device also has the capabilites of monitoring motion, skin temperature, and heart-rate in order to capture a vivid picture of the factors affecting your circadian rhythms.

Included in your kit

- Circadian Rhythm Monitor
- Mini-USB Cable
- Circadian Rhythm Analyzer Android Application

Setup

1. Plug in the mini-USB to the device. Plug the other end into a USB receptacle. An internal red light should come on indicating its charging status.
2. When the internal light turns off your device is ready for use.
3. Strap the device to your right hand so that the status LED's are oriented in the bottom right corner. This ensures proper alignment of the heart rate sensor.
4. Install the Circadian Rhythm Analyzer application on your android device. This can be found on the Android App store or downloaded online.

Syncing Date and Time

1. Open the Circadian Rhythm Analyzer application on your Android device.
2. Select "Sync Phone".
3. On your Circadian Rhythm Monitor, push the button in the upper left corner to turn on Bluetooth Low Energy connectivity. The BLE indicator light should begin flickering.
4. Your Circadian Rhythm Monitor should now show up on your phone as an available device to connect to. If it does not select "Sync" in the upper right corner of the application. Select the device.
5. On the next screen select "Sync Time".

6. When this is selected the sensors on the device are enabled and begin reading visible light, IR light, UV light, your heart rate, your skin temperature, and your motion. This data is collected and stored on board your Circadian Rhythm Monitor.
7. Turn off Bluetooth Low Energy on your Circadian Rhythm Monitor by holding down the button on the device for approximately 5 seconds until the BLE indicator light turns solid, then release the button. Turning off BLE on your device will provide you with better battery life.

Sync Data Collected Data

1. Open the Circadian Rhythm Analyzer application on your Android device.
2. Select “Sync Phone”.
3. On your Circadian Rhythm Monitor, push the button in the upper left corner to turn on Bluetooth Low Energy connectivity. The BLE indicator light should begin flickering.
4. Your Circadian Rhythm Monitor should now show up on your phone as an available device to connect to. If it does not select “Sync” in the upper right corner of the application. Select the device.
5. On the next screen select “Sync Data”.
6. Data will begin transmission from the Circadian Rhythm Monitor to your Android device. Progress is shown on the bottom of the screen, please do not disconnect your device during this process. As data is collected a CSV (comma-separated value) file is generated and stored on your phone with the title set to the current date and time of your Android device. These CSV files can be used in several ways.
    a. CSV files can be read by the Circadian Rhythm Analyzer application.
    b. CSV files can be emailed and later viewed on a computer with a spreadsheet program.
    c. CSV files can be shared with your treating physician for further analysis.
7. Turn off Bluetooth Low Energy on your Circadian Rhythm Monitor by holding down the button on the device for approximately 5 seconds until the BLE indicator light turns solid, then release the button. Turning off BLE on your device will provide you with better battery life.

Viewing Real Time Data

1. Open the Circadian Rhythm Analyzer application on your Android device.
2. Select “Sync Phone”.
3. On your Circadian Rhythm Monitor, push the button in the upper left corner to turn on Bluetooth Low Energy connectivity.
4. Your Circadian Rhythm Monitor should now show up on your phone as an available device to connect to. If it does not select “Sync” in the upper right corner of the application. Select the device.
5. On the next screen select “Real Time”.
6. Data will be displayed on the bottom of the same screen. Heart rate will not be available in real time due to Bluetooth Low Energy limiting connectivity in order to save power.
7. Turn off Bluetooth Low Energy on your Circadian Rhythm Monitor by holding down the button on the device for approximately 5 seconds until the BLE indicator light turns solid, then release the button. Turning off BLE on your device will provide you with better battery life.

Viewing Logged Data

1. Open the Circadian Rhythm Analyzer application on your Android device.
2. Select “Read CSV File”.
3. The available CSV files are displayed on the same screen in order of newest to oldest. CSV’s are titled according to the date on the phone when you synced your Circadian Rhythm Monitor. Select the CSV file you want to view data for.
4. On the next screen select the type of data you are interested in viewing.
5. The next screen will display a graph of all of the collected data for the type you selected in the previous screen. You can zoom, pinch, and slide the graph to change scaling of the graph. Sliding the graph left or right scrolls the data and the corresponding time the data was captured is shown on the X-axis.

6. Turn off Bluetooth Low Energy on your Circadian Rhythm Monitor by holding down the button on the device for approximately 5 seconds until the BLE indicator light turns solid, then release the button. Turning off BLE on your device will provide you with better battery life.

Charging

Your Circadian Rhythm Monitor is always running. Over time your Circadian Rhythm Monitor's battery will diminish. Charging the device on a regular basis will eliminate the need to resync the date and time. Should your device ever completely run out of battery, simply recharge it with the included mini-USB and remember to sync the new date and time using the Circadian Rhythm Analyzer application. Failure to resync the time will prevent the Circadian Rhythm Monitor from gathering data.

Appendix F - Bluetooth Low Energy UUID Table

Bluetooth Low Energy Protocol is used for the MCU on our device to communicate with a phone synchronizing with it. For debugging purposes there are Bluetooth profiles and services created for each of our sensors as well as FRAM. However our actual application only needs to worry about the timeservice and framservice Bluetooth profiles. The following list contains the UUID for each of the sensor's Bluetooth services:

ADXL345 Service UUID (accelerometerservice.c)	F000AA10-0451-4000-B000-00000000-0000
ADXL345 Data UUID	F000AA11-0451-4000-B000-00000000-0000
TMP006 Service UUID (irtempservice.c)	F000AA00-0451-4000-B000-00000000-0000
TMP006 Data UUID	F000AA01-0451-4000-B000-00000000-0000
SI1132 Service UUID (SI1132service.c)	F000AA20-0451-4000-B000-00000000-0000
SI1132 Data UUID	F000AA21-0451-4000-B000-00000000-0000
SI1143 Service UUID (bpmservice.c)	F000AA30-0451-4000-B000-00000000-0000
SI1143 Data UUID	F000AA31-0451-4000-B000-00000000-0000
FRAM Service UUID (framservice.c)	F000AA40-0451-4000-B000-00000000-0000
FRAM Service Data UUID	F000AA41-0451-4000-B000-00000000-0000
TIME Service UUID	F000DD10-0451-4000-B000-00000000-0000
TIME_MONTH_UUID	F000DD13-0451-4000-B000-00000000-0000
TIME_DAY_UUID	F000DD14-0451-4000-B000-00000000-0000
TIME_YEAR_UUID	F000DD15-0451-4000-B000-00000000-0000
TIME_HOUR_UUID	F000DD16-0451-4000-B000-00000000-0000
TIME_MIN_UUID	F000DD17-0451-4000-B000-00000000-0000

Operation:

As stated above the only profiles our application is truly dependent on are the services under the Time Service and the FRAM service. The Time service is important because when the MCU and phone connect via Bluetooth, the phone writes to the time service's UUIDs: DD10-DD17 which are month, day, year, hour, and minutes sent from the phone in hexadecimal values. Upon receiving all of these values from the phone when synchronized, the MCU begins to poll and process events related to the sensors. The FRAM service is important to the overall operation of our device because when the phone and MCU are synchronized, the FRAM service is used to offload the data that the current FRAM pointer is at.

Appendix G - Datasheets and References

CC2541 - Microcontroller
Datasheet http://www.ti.com/lit/ds/symlink/cc2541.pdf
User Guide http://www.ti.com/lit/ug/swru191f/swru191f.pdf

SI1132 - Light Sensor
Datasheet http://www.silabs.com/Support%20Documents/TechnicalDocs/Si1132.pdf

SI1143 - Light Sensor
Datasheet http://www.silabs.com/Support%20Documents/TechnicalDocs/Si114x.pdf

ADXL345 - Accelerometer
Datasheet http://www.analog.com/static/imported-files/data_sheets/ADXL345.pdf

TMP006 - IR Temperature Sensor
Datasheet http://www.ti.com/lit/ds/symlink/tmp006.pdf
User Guide http://www.ti.com/lit/ug/sbou107/sbou107.pdf
Layout and Assembly Guide http://www.ti.com/lit/ug/sbou108/sbou108.pdf

FM24V10 - FRAM
Datasheet http://www.cypress.com/?docID=48138

TPS62203 - DC-DC Step Down Buck Converter
Datasheet http://www.ti.com/lit/ds/symlink/tps62203.pdf

LM3658 - Dual Source USB/AC Li Chemistry Charger IC for Portable Applications
Datasheet http://www.ti.com/lit/ds/symlink/lm3658.pdf

Android Development Sources
Bluetooth Low Energy https://developer.android.com/guide/topics/connectivity/bluetooth-le.html
GraphView.jar is required for our project as it is our primary way of graphing collected data from a CSV file as an output http://www.android-graphview.org/
opencsv.jar is also required for our project as it is being used for CSV file creation http://opencsv.sourceforge.net/

Appendix H - CC2541 Firmware File Descriptions

The following sections details the implementation and integration of our sensor firmware and software:

SI1132 (SI1132.c & SI1132.h)

The SI1132 is used for IR, UV, and Visible light measurements. The following functions are used to make the SI1132 sensor operational.

Functions:

· void SI1132_Select(); - This function is used for the MCU to initialize an i2c communication with the SI1132 sensor.

· void SI1132_RESETFUNCTION(); - This function clears necessary SI1132 registers to 0xFF.

· void SI1132_SetCoefficients();- This function initializes parameters necessary for setting up the SI1132 such as ADC Overflow Response, and Auto Measurement mode.

· void SI1132_EnableUV(); - This function writes to the SI1132 internal memory and sets up the enabling of the UV Sensor by writing necessary configuration parameters

· void SI1132_EnableInterrupOnEverySample(); - This function initializes the parameters of the SI1132 register to enable interrupts for data sampling

· void SI1132_ProximitySensorSetup(); - This function initialize and calibrates the SI1132 sensors in high range mode. (High sensitivity ADC range)

· void SI1132_AutoMeasure(); - This function initializes and calibrates the rate at which the SI1132 is measuring data when it is on. It is set at a rate of 8 ms.

Operation:

· The SI1132 is polled every 15 minutes. Once the polling of the SI1132 has been enabled in the main MCU software code (SensorTag.c), the SI1132 readSI1132(UTCTime) function is called. The caller sends this function the current time determined by the UTC which is a 32 bit integer value. The readSI1132 function calls all the functions described above to initialize the SI1132 sensor to begin taking light measurements. Within this function call, 3 registers are read from: the UVINDEX , the ALSVISDATA, and the ALSIRDATA registers. Each register is 2 bytes long. Upon reading each of these registers, if the data is ready and i2c read access to all of them are successful, than the values are stored to a 11 byte array which is in the big endian format of

[Header: UTCmsb1: UTCmsb2:UTClsb1, UTClsb2:VisibleMSB: VisibleLSB:IRMSB:IRLSB:UVMSB:UVLSB]

This array of values is then copied into FRAM at the next available location in FRAM to store.

*Note: The Header value for any SI1132 value (UV,IR,VISIBLE) is always 2. This value is used for our Bluetooth communication between MCU and Phone, as the phone will be able to decode which data it is by the header.

SI1143 (SI1143.c & SI1143.h)

The SI1143 is used for the heart rate and BPM measurement counter. The following functions are used for the SI1143.

Functions:

· void SI1143_Select(); - This function is used for the MCU to initialize a new i2c connection between the CC2541 and the SI1143

· void SI1143_initPulseSensor(); - This function is the main firmware code that drives the SI1143 sensor. It is in charge of initializing and calibrating the sensors in the same matter as the SI1132 above. In addition to IR,ADC,

sampling rate, etc. sensor calibrations, this function is in charge of turning on the red LED and calibrating the current consumption, and brightness of the LED.

· Uint8 SI1143_ReadParameters (char target) This function is a helper function which returns the 1 byte value from the passed in register

· void SI1143_WriteParameters(char target, char parameters) - This function writes a value to the SI1143 internal registers

Operation:

· The Heart Rate BPM calculations are all done within the MCU software code (SensorTag.c). The SI1143 is polled every 15 minutes. Once an Event is triggered in the software to poll the SI1143, it is turned on. To produce the beats per minute calculation, once this sensor is polled, the current internal timer of the MCU in milliseconds is checked to see if it has run for 20 seconds. (This can be changed to another amount of time). During these 20 seconds the red LED shoots light up into the wrist where this device is worn. The IR sensors calculate the reflectance of this light that is being reflected from the blood stream. While this process is repeated for 20 seconds, for every 5 CPU cycles within this 20 seconds the data of the reflectance is being recorded and averaged, then put into a comparator array. An algorithm is then run on this array where the current index is compared to the last IR calculation and the next IR calculation. If the algorithm detects that this middle data is a valley and the other two points are peaks, the counter for the detected heartbeats is incremented. Once the 20 seconds is over, the BPMCounter is multiplied by 3 to scale it up to a minute of heart beats. This data is then copied into the 11 byte holder array which will be used to copy into FRAM. The array data holder format in big endian is shown below:

[Header: UTCmsb1: UTCmsb2:UTClsb1, UTClsb2:0:0:BPM_MSB1:BPM_MSB2:BPM_LSB1:BPM_LSB2]

*Note: The Header value for any SI1143 is always 3. This value is used for our Bluetooth communication between MCU and Phone, as the phone will be able to decode which data it is by the header.

TMP006 (hal_irtemp.c & hal_irtemp.h)

The TMP006 is used for temperature measurements for this device. The following functions are used to make the TMP006 sensor operational.

Functions:

· Void HalIRTempSelect() - This function is used for the MCU to initialize an i2c connection between the CC2541 and the TMP006 sensor.

· void HalIRTempTurnOn() – This function turns the TMP006 on and into an idle state

· void HalIRTempTurnOff() – This function turns off the TMP006 sensor

· IRTemperature_States_t HalIRTempStatus() – This function returns the state that the TMP006 sensor is in. If the Sensor is on, this function will check if data is ready to be read from the TMP006 registers.

Operation:

· The TMP006 is polled every 15 minutes. Once the polling of the TMP006 has been enabled in the main MCU software code (SensorTag.c), the TMP006 is first checked for the status of the sensor. If the sensor is off, the TMP006 is turned on. If the sensor is on, than the TMP006 is checked for the status of the data in its internal register. If the data is ready than the readIRTempData(UTCTime) function is called The calling block of code sends the current UTCTime to this function and it is processed the exact same way as in the previous sensors described above. Inside the readIRTempData function the Object Voltage (VObj) value and Ambient Temperature (Temp) values are read from the TMP006 registers. These data values are 2 byte unsigned values. Once these values are read, the data is passed into the 11 byte data holding array which is used for FRAM. The big endian format of this array is shown below:

[Header: UTCmsb1: UTCmsb2:UTClsb1, UTClsb2:0: 0: VObjLSB: VObjMSB : TempLSB : TempMSB]

*Note: The Header value for any TMP006 data is always 4. This value is used for our Bluetooth communication between MCU and Phone, as the phone will be able to decode which data it is by the header.

ADXL345 (hal_acc.c & hal_acc.h)

The ADXL345 is used to measure acceleration and movement for a user wearing the device. The following functions are used for the ADXL345 to be operational.

Functions:

- Void HalAccSelect() – This function lets the MCU initialize an i2c connection with the ADXL345
- void setRange(range_t range) – This sets and calibrates the range and sensitivity the ADXL345 will read measurements, our design uses a 16G calibration
- bool readAcc(uint8 *pBuf, int32 UTCTime) – reads X,Y,Z acceleration measurements and puts it into the FRAM data array holder.

Operation:

- The ADXL345 is polled every minute. Once the polling of the ADXL345 is enabled by the main MCU software (Sensortag.c), the function readAccData(Time) is called. This function takes in the time passed by the caller and processes it like the other functions. Inside the readAccData function readAcc is called which sets up the ADXL345 sensor by initializing the i2c connection and calibrating its sensor. Once calibration is done, the registers containing the X, Y, and Z measurements (2 byte values) are read into the FRAM data array holder. The array data holder format in big endian is shown below:

[Header: UTCmsb1: UTCmsb2:UTClsb1:UTClsb2: xMSB: xLSB: yMSB: yLSB: zMSB: zLSB]

*Note: The Header value for any ADXL345 is always 1. This value is used for our Bluetooth communication between MCU and Phone, as the phone will be able to decode which data it is by the header.

FM24V10 FRAM (FRAM.c & FRAM.h)

The FM24V10 IC is as external memory for our device. The fram is used to store all our sensor data, and is later read when requested by a phone device during ble synchronization.

Functions:

- void READ_CURRENT_DATA(int32 address, uint8 *FRAM_DATA) – This function reads 11 bytes of fram space (the amount our protocol takes) starting from the address given to it.This function takes in a 32 bit integer address for from from 0 to 131065, and a pointer to an array to store the read fram data into.
- void WRITE_SENSORDATA_FRAM(int32 address, uint8 data_array[]);

Operation:

The FRAM is accessed at many different times throughout device operation. The device is written to whenever the sensors of our device poll and record data. When storing data, the fram address is written to, once it is written to the fram pointer which keeps the address of the next point to write to is incremented. With our particular FRAM, there are two different pages holding 64Kb of memory. If an address goes over 64Kb depending on the page it rolls over, ie. page 1 rolls to page 2 of memory while page 2 rolls back to 1.The fram is read whenever a phone requests to offload the fram onto the phone. When the data is offloaded the offloaded memory in fram is cleared, and the pointer to the read address of the fram is incremented as well.

UTC Code(UTCTime.c & UTCTime.h)

For our device we implemented a Unix Time Clock function. This is used to calculate the exact time and date given a time from the phone during synchronization. The UTC was implemented in response to not having a RTC. A user

syncs up their phone and sends the current date and time to the MCU. From there the UTC function will then take in the parameters such as month, day, year, hour and minute and convert them into the UTC format of milliseconds since 1970. This time is then saved on the MCU where it is used to determine the exact time and date polling of data occurred.

Appendix I - Android Application File Descriptions

Understanding the fundamentals of Bluetooth Low Energy is paramount in creating a successful BLE enabled Android application. The intention of this section of the report is to explain the details of the major components of each *.java file. It is assumed that the reader has basic understanding of the BLE stack and a background in java and android development. Also note that “data-type” is representative of all of the different types of data we are collecting from our wrist device (visible light, IR light, UV light, motion, BPM, temperature, etc).

“data-type”_GRAPHER.java

These sections of code bring in the filename selected from the previous intent. It uses that filename to access the file of the same name saved on an external storage. The file (which will always be a CSV) is opened and is parsed line by line. The parser is looking for only data that is of interest to the user, which is determined in the previous intent. For example in ADXL_GRAPHER.java the parser is only looking for data that is from the “Accelerometer”. For each piece of data collected as a GraphViewData type is created and stored in an ArrayList for convenience. The ArrayList is later used to plot a graph. This section of code is associated with grapher_layout.xml

“data-type”_Obj.java

These sections of code use the raw data extracted from the FRAM/Sensors (as an input) and parse through the data to create real, meaningful data (as an output) that can be analyze and/or graphed later. This output changes for each “data-type”. This section of code also converts UTC into a meaningful Date-Time format for the user.

csv_data_selector.java

This section of code is rather simple as it pulls the selected CSV filename from the previous intent and then lets a user specify which type of data in the previously selected CSV that they would like to have graphed. This section of code is associated with csv_dataviews.xml

csv_select.java

This is the starting point for the entire application. The method “onClick” has two outcomes depending on the interaction with the user. If “Sync Phone” is clicked then a new intent is made to sync the device with the Android Device. If “Read CSV File” is clicked, An ArrayList is created in GetFiles containing all *.csv files on the Android device. That list is then displayed on the screen. When a user selects a CSV file from the list onItemClick saves that selected file name and passes that into a new intent. This section of code is associated with csv_select.xml

DeviceScanActivity.java

This section of code is where the BLE connection is handled. Major code sections discussed as an overview below.

onCreate: It begins by creating a handler that will handle of of the available BLE devices within range of the Android device. The code also performs the necessary checks to make sure the Android device is capable of BLE as well as standard Bluetooth.

onListItemClick: Creates a new intent which passes along the selected device name and address so that the new intent can pair to the selected device without having to rescan.

onOptionsItemSelected: Is written so that the user may select to restart or stop a scan.

LeDeviceListAdapter: This holds the BLE devices that were found during scanning and then add them to an ArrayList of available BLE device. This ArrayList is used to display the BLE devices.

This section of code is associated with itemlist_device.xml

GraphViewData.java
This section of code is very simple. The object is used to hold data as an x-y coordinate pair so that it may be later plotted using GraphView.

InitOptionSelect.java
This section of code handles the connection to specific UUID addresses being used on the device. It also handles the syncing of time to the device, syncing data from the device, and streaming data from the device in “real time”. The intent receives the previously selected device name and address and maintains the connection to the selected device. It also performs various error handling throughout.

sortData: Takes the input “data” and parses through it to determine its header, data fields, and time. Time is then converted from UTC to a meaningful value. Using the header bit, the code determines which type of data it is (light, motion, etc.) and calls the relevant “data-type”_Obj to convert the raw data into meaningful data. If all data checks out then it is added to the end of the CSV file (created in onCreate) and, if “Real Time” is selected, updates the corresponding value on the display.

getDateCurrentTimeZone: Used to convert from UTC into meaningful Date and Time.

onCreate: Sets up and maintains BLE connection and parameters. It also determines actions from buttonclicks.

BluetoothGattCharacteristic and readCharacteristicValue: Necessary BLE methods used for connection and reading data from the device.

real_time: Reads the values of specific BLE characteristics (motion, light and temperature).

sync_counter: Reads the value of the FRAM BLE characteristic, sorts the value based on the header, and increments a counter to show the progress.

sync_timer: Reads the value of the each of the respective time BLE characteristics (month, day, year, hour, minute).

This section of code is associated with init_option_select.xml

AndroidManifest.xml
The main purpose of this section is to include the following lines of code to enable Bluetooth and BLE permissions for the Android application.

- <uses-feature android:name="android.hardware.bluetooth_le" android:required="true"/>
- <uses-permission android:name="android.permission.BLUETOOTH"/>
- <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
- <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

Appendix J - CC2541 Firmware Source Code

FRAM.H

```
uint8 READFRAM(int32 address);
void WRITE_FRAM(int32 address, uint8 data);
void WRITE_SENSORDATA_FRAM(int32 address, uint8 data_array[]);
void READ_CURRENT_DATA(int32 address, uint8 *FRAM_DATA);
```

FRAM. C

```
#include "hal_i2c.h"
#include "stdio.h"
#define BYTE1 1
#define BYTE2 2

uint8 READFRAM(int32 address){

  uint8 i2c_pg_addr = 0x00;
  uint8 address_MSB = (address & 0xff00) >> 8;
  uint8 address_LSB = (address & 0x00ff);
  uint8 registerPointer = 0x00;

  if(address<65536){ //pgselect 0
    i2c_pg_addr = 0x50; //pg0

    HalI2CInit(0x50, i2cClock_533KHZ);

  }else{
    i2c_pg_addr = 0x51; //pg0

    HalI2CInit(0x51, i2cClock_533KHZ);
  }

  READ_FRAM_FROM_ADDRESS(BYTE1, &i2c_pg_addr, (uint8 *)&address_MSB, (uint8 *)&address_LSB,
(uint8 *)&registerPointer);

  return registerPointer;

}

void WRITE_FRAM(int32 address, uint8 data){

  uint8 i2c_pg_addr = 0x00;
  uint8 address_MSB = (address & 0xff00) >> 8;
  uint8 address_LSB = (address & 0x00ff);

  if(address<65536){ //pgselect 0
    i2c_pg_addr = 0x50; //pg0

    HalI2CInit(0x50, i2cClock_533KHZ);

  }else{
    i2c_pg_addr = 0x51; //pg0

    HalI2CInit(0x51, i2cClock_533KHZ);
```

```
  }

  WRITE_TO_FRAM_TEST((uint8 *)&address_MSB, (uint8 *)&address_LSB, (uint8 *)&data);


}

void WRITE_SENSORDATA_FRAM(int32 address, uint8 data_array[]){


  uint8 i2c_pg_addr = 0x00;
  uint8 address_MSB = (address & 0xff00) >> 8;
  uint8 address_LSB = (address & 0x00ff);

  if(address<65536){ //pgselect 0
    i2c_pg_addr = 0x50; //pg0

    HalI2CInit(0x50, i2cClock_533KHZ);

  }else{
    i2c_pg_addr = 0x51; //pg0

    HalI2CInit(0x51, i2cClock_533KHZ);
  }

 // void (uint8 *MSB, uint8 *LSB, uint8 data_array[])
  WRITE_SENSOR_DATA_TO_FRAM((uint8 *)&address_MSB, (uint8 *)&address_LSB, data_array);


}

void READ_CURRENT_DATA(int32 address, uint8 *FRAM_DATA){


  uint8 i2c_pg_addr = 0x00;
  uint8 address_MSB = (address & 0xff00) >> 8;
  uint8 address_LSB = (address & 0x00ff);
  uint8 registerPointer = 0x00;

  if(address<65536){ //pgselect 0
    i2c_pg_addr = 0x50; //pg0

    HalI2CInit(0x50, i2cClock_533KHZ);

  }else{
    i2c_pg_addr = 0x51; //pg0

    HalI2CInit(0x51, i2cClock_533KHZ);
  }

  READ_FRAM_FROM_ADDRESS(BYTE1, &i2c_pg_addr, (uint8 *)&address_MSB, (uint8 *)&address_LSB,
(uint8 *)&registerPointer);

  FRAM_DATA[0] = registerPointer;

  address_LSB++;
  READ_FRAM_FROM_ADDRESS(BYTE1, &i2c_pg_addr, (uint8 *)&address_MSB, (uint8 *)&address_LSB,
(uint8 *)&registerPointer);

  FRAM_DATA[1] = registerPointer;
```

```
    address_LSB++;
  READ_FRAM_FROM_ADDRESS(BYTE1, &i2c_pg_addr, (uint8 *)&address_MSB, (uint8 *)&address_LSB,
(uint8 *)&registerPointer);

  FRAM_DATA[2] = registerPointer;

    address_LSB++;
  READ_FRAM_FROM_ADDRESS(BYTE1, &i2c_pg_addr, (uint8 *)&address_MSB, (uint8 *)&address_LSB,
(uint8 *)&registerPointer);

  FRAM_DATA[3] = registerPointer;

    address_LSB++;
  READ_FRAM_FROM_ADDRESS(BYTE1, &i2c_pg_addr, (uint8 *)&address_MSB, (uint8 *)&address_LSB,
(uint8 *)&registerPointer);

  FRAM_DATA[4] = registerPointer;

    address_LSB++;
  READ_FRAM_FROM_ADDRESS(BYTE1, &i2c_pg_addr, (uint8 *)&address_MSB, (uint8 *)&address_LSB,
(uint8 *)&registerPointer);

  FRAM_DATA[5] = registerPointer;

    address_LSB++;
  READ_FRAM_FROM_ADDRESS(BYTE1, &i2c_pg_addr, (uint8 *)&address_MSB, (uint8 *)&address_LSB,
(uint8 *)&registerPointer);

  FRAM_DATA[6] = registerPointer;

    address_LSB++;
  READ_FRAM_FROM_ADDRESS(BYTE1, &i2c_pg_addr, (uint8 *)&address_MSB, (uint8 *)&address_LSB,
(uint8 *)&registerPointer);

  FRAM_DATA[7] = registerPointer;

    address_LSB++;
  READ_FRAM_FROM_ADDRESS(BYTE1, &i2c_pg_addr, (uint8 *)&address_MSB, (uint8 *)&address_LSB,
(uint8 *)&registerPointer);

  FRAM_DATA[8] = registerPointer;

    address_LSB++;
  READ_FRAM_FROM_ADDRESS(BYTE1, &i2c_pg_addr, (uint8 *)&address_MSB, (uint8 *)&address_LSB,
(uint8 *)&registerPointer);

  FRAM_DATA[9] = registerPointer;

    address_LSB++;
  READ_FRAM_FROM_ADDRESS(BYTE1, &i2c_pg_addr, (uint8 *)&address_MSB, (uint8 *)&address_LSB,
(uint8 *)&registerPointer);

  FRAM_DATA[10] = registerPointer;

}
```

SI1132.H

```
#ifndef SI1132_H_
#define SI1132_H_
#endif /* SI1132_H_ */

/* COMMANDS */
#define SI1132_PARAM_QUERY 0x80
#define SI1132_PARAM_SET 0xA0
#define SI1132_NOP 0x0
#define SI1132_RESET    0x01
#define SI1132_BUSADDR    0x02
#define SI1132_PS_FORCE    0x05
#define SI1132_ALS_FORCE    0x06
#define SI1132_PSALS_FORCE    0x07
#define SI1132_PS_PAUSE    0x09
#define SI1132_ALS_PAUSE    0x0A
#define SI1132_PSALS_PAUSE    0xB
#define SI1132_PS_AUTO    0x0D
#define SI1132_ALS_AUTO   0x0E
#define SI1132_PSALS_AUTO 0x0F
#define SI1132_GET_CAL    0x12

/* Parameters */
#define SI1132_PARAM_I2CADDR 0x00
#define SI1132_PARAM_CHLIST   0x01
#define SI1132_PARAM_CHLIST_ENUV 0x80
#define SI1132_PARAM_CHLIST_ENAUX 0x40
#define SI1132_PARAM_CHLIST_ENALSIR 0x20
#define SI1132_PARAM_CHLIST_ENALSVIS 0x10
#define SI1132_PARAM_CHLIST_ENPS1 0x01
#define SI1132_PARAM_CHLIST_ENPS2 0x02
#define SI1132_PARAM_CHLIST_ENPS3 0x04

#define SI1132_PARAM_PSLED12SEL   0x02
#define SI1132_PARAM_PSLED12SEL_PS2NONE 0x00
#define SI1132_PARAM_PSLED12SEL_PS2LED1 0x10
#define SI1132_PARAM_PSLED12SEL_PS2LED2 0x20
#define SI1132_PARAM_PSLED12SEL_PS2LED3 0x40
#define SI1132_PARAM_PSLED12SEL_PS1NONE 0x00
#define SI1132_PARAM_PSLED12SEL_PS1LED1 0x01
#define SI1132_PARAM_PSLED12SEL_PS1LED2 0x02
#define SI1132_PARAM_PSLED12SEL_PS1LED3 0x04

#define SI1132_PARAM_PSLED3SEL   0x03
#define SI1132_PARAM_PSENCODE   0x05
#define SI1132_PARAM_ALSENCODE  0x06

#define SI1132_PARAM_PS1ADCMUX   0x07
#define SI1132_PARAM_PS2ADCMUX   0x08
#define SI1132_PARAM_PS3ADCMUX   0x09
#define SI1132_PARAM_PSADCOUNTER   0x0A
#define SI1132_PARAM_PSADCGAIN 0x0B
#define SI1132_PARAM_PSADCMISC 0x0C
#define SI1132_PARAM_PSADCMISC_RANGE 0x20
#define SI1132_PARAM_PSADCMISC_PSMODE 0x04

#define SI1132_PARAM_ALSIRADCMUX   0x0E
#define SI1132_PARAM_AUXADCMUX   0x0F

#define SI1132_PARAM_ALSVISADCOUNTER   0x10
#define SI1132_PARAM_ALSVISADCGAIN 0x11
```

```
#define SI1132_PARAM_ALSVISADCMISC 0x12
#define SI1132_PARAM_ALSVISADCMISC_VISRANGE 0x20

#define SI1132_PARAM_ALSIRADCOUNTER   0x1D
#define SI1132_PARAM_ALSIRADCGAIN 0x1E
#define SI1132_PARAM_ALSIRADCMISC 0x1F
#define SI1132_PARAM_ALSIRADCMISC_RANGE 0x20

#define SI1132_PARAM_ADCCOUNTER_511CLK 0x70

#define SI1132_PARAM_ADCMUX_SMALLIR  0x00
#define SI1132_PARAM_ADCMUX_LARGEIR  0x03



/* REGISTERS */
#define SI1132_REG_PARTID  0x00
#define SI1132_REG_REVID  0x01
#define SI1132_REG_SEQID  0x02

#define SI1132_REG_INTCFG  0x03
#define SI1132_REG_INTCFG_INTOE 0x01
#define SI1132_REG_INTCFG_INTMODE 0x02

#define SI1132_REG_IRQEN  0x04
#define SI1132_REG_IRQEN_ALSEVERYSAMPLE 0x01
#define SI1132_REG_IRQEN_PS1EVERYSAMPLE 0x04
#define SI1132_REG_IRQEN_PS2EVERYSAMPLE 0x08
#define SI1132_REG_IRQEN_PS3EVERYSAMPLE 0x10


#define SI1132_REG_IRQMODE1 0x05
#define SI1132_REG_IRQMODE2 0x06

#define SI1132_REG_HWKEY  0x07
#define SI1132_REG_MEASRATE0 0x08
#define SI1132_REG_MEASRATE1  0x09
#define SI1132_REG_PSRATE  0x0A
#define SI1132_REG_PSLED21  0x0F
#define SI1132_REG_PSLED3  0x10
#define SI1132_REG_UCOEFF0  0x13
#define SI1132_REG_UCOEFF1  0x14
#define SI1132_REG_UCOEFF2  0x15
#define SI1132_REG_UCOEFF3  0x16
#define SI1132_REG_PARAMWR  0x17
#define SI1132_REG_COMMAND  0x18
#define SI1132_REG_RESPONSE  0x20
#define SI1132_REG_IRQSTAT  0x21
#define SI1132_REG_IRQSTAT_ALS  0x01

#define SI1132_REG_ALSVISDATA0 0x22
#define SI1132_REG_ALSVISDATA1 0x23
#define SI1132_REG_ALSIRDATA0 0x24
#define SI1132_REG_ALSIRDATA1 0x25
#define SI1132_REG_PS1DATA0 0x26
#define SI1132_REG_PS1DATA1 0x27
#define SI1132_REG_PS2DATA0 0x28
#define SI1132_REG_PS2DATA1 0x29
#define SI1132_REG_PS3DATA0 0x2A
#define SI1132_REG_PS3DATA1 0x2B
```

```
#define SI1132_REG_UVINDEX0 0x2C
#define SI1132_REG_UVINDEX1 0x2D
#define SI1132_REG_PARAMRD 0x2E
#define SI1132_REG_CHIPSTAT 0x30

#define SI1132_I2C_ADDRESS 0x60
```

SI1132.C

```
#include "SI1132.h"
#include "hal_drivers.h"

#include "bcomdef.h"
#include "OSAL.h"
#include "OSAL_PwrMgr.h"

#include "OnBoard.h"
#include "hal_adc.h"
#include "hal_led.h"
#include "hal_keys.h"
#include "hal_i2c.h"

#include "gatt.h"
#include "hci.h"

#include "gapgattserver.h"
#include "gattservapp.h"

#if defined ( PLUS_BROADCASTER )
  #include "peripheralBroadcaster.h"
#else
  #include "peripheral.h"
#endif

#include "gapbondmgr.h"

#if defined FEATURE_OAD
  #include "oad.h"
  #include "oad_target.h"
#endif

// Services
#include "st_util.h"
#include "devinfoservice-st.h"
#include "irtempservice.h"
#include "accelerometerservice.h"
#include "humidityservice.h"
#include "magnetometerservice.h"
#include "barometerservice.h"
#include "gyroservice.h"
#include "testservice.h"
#include "simplekeys.h"
#include "ccservice.h"

// Sensor drivers
#include "sensorTag.h"
```

```
#include "hal_sensor.h"

#include "hal_irtemp.h"
#include "hal_acc.h"
#include "hal_humi.h"
#include "hal_mag.h"
#include "hal_bar.h"
#include "hal_gyro.h"

#include "stdio.h"


#define BYTE1 1
#define BYTE2 2

void SI1132_WriteParameters(char target, char parameters);


void SI1132_Select(){


HalI2CInit(SI1132_I2C_ADDRESS, i2cClock_533KHZ);

}

int SI1132_begin() {

  uint8 registerPointer;

 // SI1132_Select();

       bool success = HalSensorReadReg(SI1132_REG_PARTID, (uint8 *)&registerPointer, BYTE1);

        printf("\nREGISTERPOINTER:%x\n", registerPointer);

        if(success){
          if (registerPointer == 0x32)
            printf("1");
            return 1; // look for SI1132
        }


       //DEVICE FOUND Reset SI1132 Registers

        printf("0");
       return 0;
}


void SI1132_RESETFUNCTION() {

  uint8 value_pointer = 0;

  HalSensorWriteReg(SI1132_REG_MEASRATE0, &value_pointer, BYTE1);
  HalSensorWriteReg(SI1132_REG_MEASRATE1, &value_pointer, BYTE1);
  HalSensorWriteReg(SI1132_REG_IRQEN, &value_pointer, BYTE1);
  HalSensorWriteReg(SI1132_REG_IRQMODE1, &value_pointer, BYTE1);
  HalSensorWriteReg(SI1132_REG_IRQMODE2, &value_pointer, BYTE1);
  HalSensorWriteReg(SI1132_REG_INTCFG, &value_pointer, BYTE1);
```

```
  value_pointer = 0xFF;

  HalSensorWriteReg(SI1132_REG_IRQSTAT, &value_pointer, BYTE1);

  value_pointer = SI1132_RESET;


  HalSensorWriteReg(SI1132_REG_COMMAND, &value_pointer, BYTE1);
       //__delay_cycles(10000); // delay program execution for 1000 cycles

  value_pointer = 0x17;

  HalSensorWriteReg(SI1132_REG_HWKEY, &value_pointer, BYTE1);
       //__delay_cycles(10000); // delay program execution for 1000 cycles
}

void SI1132_SetCoefficients() {

        uint8 value_pointer;

        value_pointer = 0x29;

       HalSensorWriteReg(SI1132_REG_UCOEFF0, &value_pointer, BYTE1);

        value_pointer = 0x89;

       HalSensorWriteReg(SI1132_REG_UCOEFF1, &value_pointer, BYTE1);

        value_pointer = 0x02;

       HalSensorWriteReg(SI1132_REG_UCOEFF2, &value_pointer, BYTE1);

        value_pointer = 0x00;

       HalSensorWriteReg(SI1132_REG_UCOEFF3, &value_pointer, BYTE1);
}

void SI1132_EnableUV(void) {

       SI1132_WriteParameters(SI1132_PARAM_CHLIST,
                      (SI1132_PARAM_CHLIST_ENUV | SI1132_PARAM_CHLIST_ENALSIR
                                     | SI1132_PARAM_CHLIST_ENALSVIS |
SI1132_PARAM_CHLIST_ENPS1));
}

void SI1132_WriteParameters(char target, char parameters) {

  uint8 registerPointer;

  uint8 value_pointer;

  value_pointer = parameters;

       HalSensorWriteReg(SI1132_REG_PARAMWR, &value_pointer, BYTE1); // Pass parameter x to
the mailbox register (host to sequencer)

        value_pointer = (target | SI1132_PARAM_SET);

       HalSensorWriteReg(SI1132_REG_COMMAND, &value_pointer ,BYTE1 ); //Sets parameter
pointed by target with value just placed into mailbox register
```

```
       HalSensorReadReg(SI1132_REG_PARAMRD,(uint8 *)&registerPointer, BYTE1); // Read from
the mailbox register (sequencer to host) (required to clear)
}

uint8 SI1132_ReadParameters(char target) {
   uint8 registerPointer;
     uint8 value_pointer;

     value_pointer = (target | SI1132_PARAM_QUERY);

       HalSensorWriteReg(SI1132_REG_COMMAND, &value_pointer , BYTE1);
       HalSensorReadReg(SI1132_REG_PARAMRD,(uint8 *)&registerPointer, BYTE1);

        return  registerPointer;
}

void SI1132_EnableInterrupOnEverySample(void) {

  uint8 value_pointer;

  value_pointer = SI1132_REG_INTCFG_INTOE;

       HalSensorWriteReg(SI1132_REG_INTCFG, &value_pointer, BYTE1);

        value_pointer = SI1132_REG_IRQEN_ALSEVERYSAMPLE;

       HalSensorWriteReg(SI1132_REG_IRQEN, &value_pointer, BYTE1);
}

void SI1132_ProximitySensorSetup(void) {

  uint8 value_pointer;

  value_pointer = 0x03;
       // program LED current
       HalSensorWriteReg(SI1132_REG_PSLED21, &value_pointer, BYTE1); // 20mA for LED 1 only
       SI1132_WriteParameters(SI1132_PARAM_PS1ADCMUX,
                     SI1132_PARAM_ADCMUX_LARGEIR);
       // prox sensor #1 uses LED #1
       SI1132_WriteParameters(SI1132_PARAM_PSLED12SEL,
                     SI1132_PARAM_PSLED12SEL_PS1LED1);
       // fastest clocks, clock div 1
       SI1132_WriteParameters(SI1132_PARAM_PSADCGAIN, 0);
       // take 511 clocks to measure
       SI1132_WriteParameters(SI1132_PARAM_PSADCOUNTER,
                     SI1132_PARAM_ADCCOUNTER_511CLK);
       // in prox mode, high range
       SI1132_WriteParameters(SI1132_PARAM_PSADCMISC,
                     (SI1132_PARAM_PSADCMISC_RANGE | SI1132_PARAM_PSADCMISC_PSMODE));

       SI1132_WriteParameters(SI1132_PARAM_ALSIRADCMUX,
                     SI1132_PARAM_ADCMUX_SMALLIR);
       // fastest clocks, clock div 1
       SI1132_WriteParameters(SI1132_PARAM_ALSIRADCGAIN, 0);
       // take 511 clocks to measure
       SI1132_WriteParameters(SI1132_PARAM_ALSIRADCOUNTER,
                     SI1132_PARAM_ADCCOUNTER_511CLK);
       // in high range mode
       SI1132_WriteParameters(SI1132_PARAM_ALSIRADCMISC,
                     SI1132_PARAM_ALSIRADCMISC_RANGE);
```

```
        // fastest clocks, clock div 1
        SI1132_WriteParameters(SI1132_PARAM_ALSVISADCGAIN, 0);
        // take 511 clocks to measure
        SI1132_WriteParameters(SI1132_PARAM_ALSVISADCOUNTER,
                        SI1132_PARAM_ADCCOUNTER_511CLK);
        // in high range mode (not normal signal)
        SI1132_WriteParameters(SI1132_PARAM_ALSVISADCMISC,
                        SI1132_PARAM_ALSVISADCMISC_VISRANGE);
}

void SI1132_AutoMeasure(void) {
        // measurement rate for auto

  uint8 value_pointer;

  value_pointer = 0xFF;

        HalSensorWriteReg(SI1132_REG_MEASRATE0, &value_pointer,BYTE1); // 255 * 31.25uS = 8ms
        // auto run

         value_pointer = SI1132_PSALS_AUTO;

        HalSensorWriteReg(SI1132_REG_COMMAND, &value_pointer,BYTE1);
}

uint16 SI1132_ReadUV(void) {

uint16 registerPointer;

HalSensorReadReg(SI1132_REG_UVINDEX0,(uint8 *)&registerPointer, BYTE2);

        return registerPointer;
}

uint16 SI1132_ReadVisible(void) {

  uint16 registerPointer;
  HalSensorReadReg(SI1132_REG_ALSVISDATA0,(uint8 *)&registerPointer, BYTE2);

  return registerPointer;
}

uint16 SI1132_ReadIR(void) {

  uint16 registerPointer;

  HalSensorReadReg(SI1132_REG_ALSIRDATA0,(uint8 *)&registerPointer, BYTE2);

        return registerPointer;


}

uint16 SI1132_ReadProx(void) {

  uint16 registerPointer;

        HalSensorReadReg(SI1132_REG_PS1DATA0,(uint8 *)&registerPointer, BYTE2);
         return registerPointer;
}
```

```
bool readSI1132(uint8 *pBuf, int32 UTCTime){

   uint16 visible;
  uint16 IR;
  uint16 UV;
  bool success;

  SI1132_Select();

  SI1132_RESETFUNCTION();
       SI1132_SetCoefficients();
        SI1132_EnableUV();
        SI1132_EnableInterrupOnEverySample();
        SI1132_ProximitySensorSetup();
        SI1132_AutoMeasure();

  ST_HAL_DELAY(180);

  // Read the three registers
  success = HalSensorReadReg(SI1132_REG_UVINDEX0,(uint8 *)&UV, BYTE2);
  //success = HalSensorReadReg( ACC_REG_ADDR_XOUT_H, &x, sizeof(x));
  if (success)
  {
    success = HalSensorReadReg(SI1132_REG_ALSVISDATA0,(uint8 *)&visible, BYTE2);
    if (success)
    {
       success = HalSensorReadReg(SI1132_REG_ALSIRDATA0,(uint8 *)&IR, BYTE2);
    }
  }

  if (success)
  {



    // Valid data

    uint8 LSB_byte2 = (UTCTime & 0x000000ff);
    uint8 LSB_byte1 = (UTCTime & 0x0000ff00) >> 8;
    uint8 MSB_byte2 = (UTCTime & 0x00ff0000) >> 16;
    uint8 MSB_byte1 = (UTCTime & 0xff000000) >> 24;

    pBuf[0] = 2;
    pBuf[1] = MSB_byte1;
    pBuf[2] = MSB_byte2;
    pBuf[3] = LSB_byte1;
    pBuf[4] = LSB_byte2;

    pBuf[5] = HI_UINT16( visible );
    pBuf[6] = LO_UINT16( visible );
    pBuf[7] = HI_UINT16( IR );
    pBuf[8] = LO_UINT16( IR );
    pBuf[9] = HI_UINT16( UV );
    pBuf[10] = LO_UINT16( UV );



  }
```

```
  //Turn off sensors

 //SI1132_WriteParameters(SI1132_PARAM_CHLIST, 0x00);


  return success;
  }
```

SI1143.H

```
#ifndef SI1143_H_
#define SI1143_H_

#endif /* SI1143_H_ */


#include "bcomdef.h"
#include "OSAL.h"
#include "OSAL_PwrMgr.h"

#include "OnBoard.h"
#include "hal_adc.h"
#include "hal_led.h"
#include "hal_keys.h"
#include "hal_i2c.h"

#include "gatt.h"
#include "hci.h"


#include "gapgattserver.h"
#include "gattservapp.h"

#if defined ( PLUS_BROADCASTER )
  #include "peripheralBroadcaster.h"
#else
  #include "peripheral.h"
#endif

#include "gapbondmgr.h"

#if defined FEATURE_OAD
  #include "oad.h"
  #include "oad_target.h"
#endif

// Services
#include "st_util.h"
#include "devinfoservice-st.h"
#include "irtempservice.h"
#include "accelerometerservice.h"
#include "SI1132service.h"
#include "humidityservice.h"
#include "magnetometerservice.h"
#include "testwritetimeservice.h"
#include "barometerservice.h"
#include "gyroservice.h"
```

```
#include "testservice.h"
#include "simplekeys.h"
#include "ccservice.h"

// Sensor drivers
#include "sensorTag.h"
#include "hal_sensor.h"

#include "hal_irtemp.h"
#include "hal_acc.h"
#include "hal_humi.h"
#include "hal_mag.h"
#include "hal_bar.h"
#include "hal_gyro.h"
#include "SI1132.h"
#include "UTCTime.h"

/*COMMAND REGISTERS*/

#define NOP_cmd          0x00  // Forces a zero into the RESPONSE register
#define RESET_cmd        0x01     // Performs a software reset of the firmware
#define BUSADDR_cmd      0x02    // Modifies I2C address
#define PS_FORCE_cmd     0x05     // Forces a single PS measurement
#define PSALS_FORCE_cmd  0x07   // Forces a single PS and ALS measurement
#define PS_PAUSE_cmd     0x09    // Pauses autonomous PS
#define ALS_PAUSE_cmd    0x0A   // Pauses autonomous ALS
#define PSALS_PAUSE_cmd  0x0B   // Pauses PS and ALS
#define PS_AUTO_cmd      0x0C   // Starts/Restarts an autonomous PS Loop
#define ALS_AUTO_cmd     0x0D    // Starts/Restarts an autonomous ALS Loop
#define PSALS_AUTO_Cmd   0x0F    // Starts/Restarts autonomous ALS and PS loop

/* PARAMS */

#define SI1143_PARAM_SET 0xA0
#define SI1143_PARAM_QUERY 0x80

#define SI1143_PARAM_I2C_ADDR            0x00
#define SI1143_PARAM_CH_LIST             0x01
#define SI1143_PARAM_PSLED12_SELECT      0x02
#define SI1143_PARAM_PSLED3_SELECT       0x03
#define SI1143_PARAM_FILTER_EN           0x04
#define SI1143_PARAM_PS_ENCODING         0x05
#define SI1143_PARAM_ALS_ENCODING        0x06
#define SI1143_PARAM_PS1_ADCMUX          0x07
#define SI1143_PARAM_PS2_ADCMUX          0x08
#define SI1143_PARAM_PS3_ADCMUX          0x09
#define SI1143_PARAM_PS_ADC_COUNTER      0x0A
#define SI1143_PARAM_PS_ADC_CLKDIV       0x0B
#define SI1143_PARAM_PS_ADC_GAIN         0x0B
#define SI1143_PARAM_PS_ADC_MISC         0x0C
#define SI1143_PARAM_ALS1_ADC_MUX        0x0D
#define SI1143_PARAM_ALS2_ADC_MUX        0x0E
#define SI1143_PARAM_ALS3_ADC_MUX        0x0F
#define SI1143_PARAM_ALSVIS_ADC_COUNTER  0x10
#define SI1143_PARAM_ALSVIS_ADC_CLKDIV   0x11
#define SI1143_PARAM_ALSVIS_ADC_GAIN     0x11
#define SI1143_PARAM_ALSVIS_ADC_MISC     0x12
#define SI1143_PARAM_ALS_HYST            0x16
#define SI1143_PARAM_PS_HYST             0x17
#define SI1143_PARAM_PS_HISTORY          0x18
```

```
#define SI1143_PARAM_ALS_HISTORY         0x19
#define SI1143_PARAM_ADC_OFFSET          0x1A
#define SI1143_PARAM_SLEEP_CTRL          0x1B
#define SI1143_PARAM_LED_RECOVERY        0x1C
#define SI1143_PARAM_ALSIR_ADC_COUNTER   0x1D
#define SI1143_PARAM_ALSIR_ADC_CLKDIV    0x1E
#define SI1143_PARAM_ALSIR_ADC_GAIN      0x1E
#define SI1143_PARAM_ALSIR_ADC_MISC      0x1F

/* COMMANDS */
#define SI1143_REG_PART_ID                 0x00
#define SI1143_REG_REV_ID                  0x01
#define SI1143_REG_SEQ_ID                  0x02
#define SI1143_REG_INT_CFG                 0x03
#define SI1143_REG_IRQ_ENABLE                      0x04
#define SI1143_REG_IRQ_MODE1               0x05
#define SI1143_REG_IRQ_MODE2               0x06
#define SI1143_REG_HW_KEY                  0x07
#define SI1143_REG_MEAS_RATE               0x08
#define SI1143_REG_ALS_RATE                0x09
#define SI1143_REG_PS_RATE                 0x0A
#define SI1143_REG_ALS_LOW_TH0                     0x0B
#define SI1143_REG_ALS_LOW_TH1                     0x0C
#define SI1143_REG_ALS_HI_TH0                      0x0D
#define SI1143_REG_ALS_HI_TH1                      0x0E
#define SI1143_REG_PS_LED21                0x0F
#define SI1143_REG_PS_LED3                 0x10
#define SI1143_REG_PS1_TH0                 0x11
#define SI1143_REG_PS1_TH1                 0x12
#define SI1143_REG_PS2_TH0                 0x13
#define SI1143_REG_PS2_TH1                 0x14
#define SI1143_REG_PS3_TH0                 0x15
#define SI1143_REG_PS3_TH1                 0x16
#define SI1143_REG_PARAM_WR                0x17
#define SI1143_REG_COMMAND                 0x18
#define SI1143_REG_RESPONSE                0x20
#define SI1143_REG_IRQ_STATUS                      0x21
#define SI1143_REG_ALS_VIS_DATA0           0x22
#define SI1143_REG_ALS_VIS_DATA1           0x23
#define SI1143_REG_ALS_IR_DATA0            0x24
#define SI1143_REG_ALS_IR_DATA1            0x25
#define SI1143_REG_PS1_DATA0               0x26
#define SI1143_REG_PS1_DATA1               0x27
#define SI1143_REG_PS2_DATA0               0x28
#define SI1143_REG_PS2_DATA1               0x29
#define SI1143_REG_PS3_DATA0               0x3A
#define SI1143_REG_PS3_DATA1               0x2B
#define SI1143_REG_AUX_DATA0               0x2C
#define SI1143_REG_AUX_DATA1               0x2D
#define SI1143_REG_PARAM_RD                0x2E
#define SI1143_REG_CHIP_STAT               0x30
#define SI1143_REG_ANA_IN_KEY1                     0x3B
#define SI1143_REG_ANA_IN_KEY2                     0x3C
#define SI1143_REG_ANA_IN_KEY3                     0x3D
#define SI1143_REG_ANA_IN_KEY4                     0x3E


#define SI1143_I2C_ADDRESS 0x5A

void SI1143_WriteParameters(char target, char parameters);
```

```
void SI1143_initPulseSensor(void);
void SI1143_readPulseSensor(void);
void SI1143_Select(void);
int SI1143_begin(void);
uint8 SI1143_ReadParameters(char target);
void SI1143_WriteParameters(char target, char parameters);
void SI1143_initPulseSensor(void);
float smooth(float data, float filterVal, float smoothedVal);
uint8 SI1143_ReadRESPONSE(void);
uint16 SI1143_ReadVisible(void);
uint16 SI1143_ReadIR(void);
uint16 SI1143_ReadPS1(void);
uint16 SI1143_ReadPS2(void);
uint16 SI1143_ReadPS3(void);
uint16 SI1143_ReadAUX(void);
void fetchData(void);
void fetchLedData(void);
bool constrain(int x, int a, int b);
void SI1143_readPulseSensor(void);
```

SI1143.C

```
#include "SI1143.h"
#include "hal_drivers.h"

#include "bcomdef.h"
#include "OSAL.h"
#include "OSAL_PwrMgr.h"

#include "OnBoard.h"
#include "hal_adc.h"
#include "hal_led.h"
#include "hal_keys.h"
#include "hal_i2c.h"

#include "gatt.h"
#include "hci.h"

#include "gapgattserver.h"
#include "gattservapp.h"

#if defined ( PLUS_BROADCASTER )
  #include "peripheralBroadcaster.h"
#else
  #include "peripheral.h"
#endif

#include "gapbondmgr.h"

#if defined FEATURE_OAD
  #include "oad.h"
  #include "oad_target.h"
#endif

// Services
#include "st_util.h"
#include "devinfoservice-st.h"
```

```
#include "irtempservice.h"
#include "accelerometerservice.h"
#include "humidityservice.h"
#include "magnetometerservice.h"
#include "barometerservice.h"
#include "gyroservice.h"
#include "testservice.h"
#include "simplekeys.h"
#include "ccservice.h"

// Sensor drivers
#include "sensorTag.h"
#include "hal_sensor.h"

#include "hal_irtemp.h"
#include "hal_acc.h"
#include "hal_humi.h"
#include "hal_mag.h"
#include "hal_bar.h"
#include "hal_gyro.h"

#include "stdio.h"


#define BYTE1 1
#define BYTE2 2

// Change I2C Address to SI1143
void SI1143_Select(){


HalI2CInit(SI1143_I2C_ADDRESS, i2cClock_533KHZ);

}


int SI1143_begin() {

  uint8 registerPointer = 1;

 // SI1146_Select();

       bool success = HalSensorReadReg(SI1143_REG_PART_ID, (uint8 *)&registerPointer,
BYTE1);

        printf("\nREGISTERPOINTER:%x\n", registerPointer);

        if(success){
          if (registerPointer == 0x43)
            printf("1");
            return 1; // look for SI1145
        }


       //DEVICE FOUND Reset SI1146 Registers

        printf("0");
       return 0;
}
```

```
uint8 SI1143_ReadParameters(char target) {
   uint8 registerPointer;
     uint8 value_pointer;

     value_pointer = (target | SI1143_PARAM_QUERY);

       HalSensorWriteReg(SI1143_REG_COMMAND, &value_pointer , BYTE1);
       HalSensorReadReg(SI1143_REG_PARAM_RD,(uint8 *)&registerPointer, BYTE1);

        return  registerPointer;
}



void SI1143_WriteParameters(char target, char parameters) {

  uint8 registerPointer;

  uint8 value_pointer;

  value_pointer = parameters;

       HalSensorWriteReg(SI1143_REG_PARAM_WR, &value_pointer, BYTE1); // Pass parameter x to
the mailbox register (host to sequencer)

        value_pointer = (target | SI1143_PARAM_SET);

       HalSensorWriteReg(SI1143_REG_COMMAND, &value_pointer ,BYTE1 ); //Sets parameter
pointed by target with value just placed into mailbox register
       HalSensorReadReg(SI1143_REG_PARAM_RD,(uint8 *)&registerPointer, BYTE1); // Read from
the mailbox register (sequencer to host) (required to clear)
}


void SI1143_initPulseSensor(){

    uint8 value_pointer;

    value_pointer = 0x17;
    HalSensorWriteReg(SI1143_REG_HW_KEY, &value_pointer,BYTE1);

    value_pointer = 0x03;
    HalSensorWriteReg(SI1143_REG_INT_CFG , &value_pointer ,BYTE1);

    value_pointer = 0x10;
    HalSensorWriteReg(SI1143_REG_IRQ_ENABLE, &value_pointer,BYTE1);

    value_pointer = 0x01;
    HalSensorWriteReg(SI1143_REG_IRQ_MODE2, &value_pointer,BYTE1);

    value_pointer = 0x84;
    HalSensorWriteReg(SI1143_REG_MEAS_RATE, &value_pointer,BYTE1);

    value_pointer = 0x08;
    HalSensorWriteReg(SI1143_REG_ALS_RATE, &value_pointer,BYTE1);

    value_pointer = 0x08;
    HalSensorWriteReg(SI1143_REG_PS_RATE, &value_pointer,BYTE1);
```

```
    // Current setting for LEDs pulsed while taking readings
    // PS_LED21  Setting for LEDs 1 & 2. LED 2 is high nibble
    // each LED has 16 possible (0-F in hex) possible settings
    // read the

    value_pointer = 0x38;
    HalSensorWriteReg(SI1143_REG_PS_LED21, &value_pointer,BYTE1);

    value_pointer = 0x03;
    HalSensorWriteReg(SI1143_REG_PS_LED3, &value_pointer,BYTE1);



   /* Serial.print( "PS_LED21 = ");
    Serial.println(getReg(PulsePlug::PS_LED21), BIN);
    Serial.print("CHLIST = ");
    Serial.println(readParam(0x01), BIN);*/

    SI1143_WriteParameters(SI1143_PARAM_CH_LIST, 0x77);
    //writeParam(PulsePlug::PARAM_CH_LIST, 0x77);         // all measurements on

    // increasing PARAM_PS_ADC_GAIN will increase the LED on time and ADC window
    // you will see increase in brightness of visible LED's, ADC output, & noise
    // datasheet warns not to go beyond 4 because chip or LEDs may be damaged
    SI1143_WriteParameters(SI1143_PARAM_PS_ADC_GAIN, 0x00);
    //writeParam(PulsePlug::PARAM_PS_ADC_GAIN, 0x00);


    // You can select which LEDs are energized for each reading.
    // The settings below turn on only the LED that "normally" would be read
    // ie LED1 is pulsed and read first, then LED2 is pulsed and read etc.
    SI1143_WriteParameters(SI1143_PARAM_PSLED12_SELECT, 0x21);
    SI1143_WriteParameters(SI1143_PARAM_PSLED3_SELECT, 0x04);

   // writeParam(PulsePlug::PARAM_PSLED12_SELECT, 0x21);  // 21 = LED 2 & LED 1 (red) resp.
   // writeParam(PulsePlug::PARAM_PSLED3_SELECT, 0x04);   // 4 = LED 3 only

    // Sensors for reading the three LEDs
    // 0x03: Large IR Photodiode
    // 0x02: Visible Photodiode - cannot be read with LEDs on - just for ambient measurement
    // 0x00: Small IR Photodiode

    SI1143_WriteParameters(SI1143_PARAM_PS1_ADCMUX, 0x03);
    SI1143_WriteParameters(SI1143_PARAM_PS2_ADCMUX, 0x03);
    SI1143_WriteParameters(SI1143_PARAM_PS3_ADCMUX, 0x03);

   // writeParam(PulsePlug::PARAM_PS1_ADCMUX, 0x03);      // PS1 photodiode select
   // writeParam(PulsePlug::PARAM_PS2_ADCMUX, 0x03);      // PS2 photodiode select
   // writeParam(PulsePlug::PARAM_PS3_ADCMUX, 0x03);      // PS3 photodiode select


    SI1143_WriteParameters(SI1143_PARAM_PS_ADC_COUNTER, 0x70);

   // writeParam(PulsePlug::PARAM_PS_ADC_COUNTER, B01110000);    // B01110000 is default
    value_pointer = PSALS_AUTO_Cmd;
    HalSensorWriteReg(SI1143_REG_COMMAND, &value_pointer,BYTE1);

   // setReg(PulsePlug::COMMAND, PulsePlug::PSALS_AUTO_Cmd);     // starts an autonomous
read loop
```

```
    // Serial.println(getReg(PulsePlug::CHIP_STAT), HEX);
    //Serial.print("end init");
}

float smooth(float data, float filterVal, float smoothedVal){

    if (filterVal > 1){      // check to make sure param's are within range
        filterVal = .99;
    }
    else if (filterVal <= 0.0){
        filterVal = 0.01;
    }

    smoothedVal = (data * (1.0 - filterVal)) + (smoothedVal  *  filterVal);
    return smoothedVal;
}

uint8 SI1143_ReadRESPONSE(void) {

  uint8 registerPointer;
  HalSensorReadReg(SI1143_REG_RESPONSE,(uint8 *)&registerPointer, BYTE1);

  return registerPointer;
}

uint16 SI1143_ReadVisible(void) {

  uint16 registerPointer;
  HalSensorReadReg(SI1143_REG_ALS_VIS_DATA0,(uint8 *)&registerPointer, BYTE2);

  return registerPointer;
}

uint16 SI1143_ReadIR(void) {

  uint16 registerPointer;
  HalSensorReadReg(SI1143_REG_ALS_IR_DATA0,(uint8 *)&registerPointer, BYTE2);

  return registerPointer;
}

uint16 SI1143_ReadPS1(void) {

  uint16 registerPointer;
  HalSensorReadReg(SI1143_REG_PS1_DATA0,(uint8 *)&registerPointer, BYTE2);

  return registerPointer;
}

uint16 SI1143_ReadPS2(void) {

  uint16 registerPointer;
  HalSensorReadReg(SI1143_REG_PS2_DATA0,(uint8 *)&registerPointer, BYTE2);

  return registerPointer;
}

uint16 SI1143_ReadPS3(void) {

  uint16 registerPointer;
```

```
  HalSensorReadReg(SI1143_REG_PS3_DATA0,(uint8 *)&registerPointer, BYTE2);

  return registerPointer;
}

uint16 SI1143_ReadAUX(void) {

  uint16 registerPointer;
  HalSensorReadReg(SI1143_REG_AUX_DATA0,(uint8 *)&registerPointer, BYTE2);

  return registerPointer;
}


bool constrain(int x, int a, int b){

  if(x>=a && x<=b){
      return x;
  }

  if(x < a){
       return a;
  }

  return b;
}
```

UTCTIME.H

```
#ifndef UTCTime_H_
#define UTCTime_H_

#endif /* UTCTime_H_ */

int32 calculateUTCTime(uint8 month, uint8 day, uint8 year, uint8 hour, uint8 min);
```

UTCTIME.C

```
#include "hal_drivers.h"

int32 calculateUTCTime(uint8 month, uint8 day, uint8 year, uint8 hour, uint8 min){

  //months is months since January 0-11
  //year only has last 2 digits
  //hours is in military 0-24


  int32 secs_day = 86400;
  int32 secs_hour = 3600;
  int32 secs_min = 60;
  int32 secs_year = 31536000;

    int monthDays = 0;

  switch(month){
```

```
  case(0x01):
    monthDays = 31; //31
    break;
  case(0x02):
    monthDays = 59; //28
    break;
  case(0x03):
    monthDays = 90; //+31
    break;
  case(0x04):
    monthDays = 120;  //+30
    break;
  case(0x05):
    monthDays = 151; //+31
    break;
  case(0x06):
    monthDays = 181; //+30
    break;
  case(0x07):
    monthDays = 212; //+31
    break;
  case(0x08):
    monthDays = 242; //+30
    break;
  case(0x09):
    monthDays = 273; //+31
    break;
  case(0x0a):
    monthDays = 303; //+30
    break;
  case(0x0b):
    monthDays = 334; //+31
    break;

}



int year1 = 2000+ year;
int years_since_1970 = year1 - 1970;


int leapyears = (years_since_1970/4) - 1;



int32 yearSeconds = years_since_1970*secs_year;
int32 leapyearSeconds = leapyears*secs_day;
int32 monthSeconds = monthDays*secs_day;
int32 daySeconds = day*secs_day;

int32 test = yearSeconds+leapyearSeconds+monthSeconds+daySeconds;


int32 hourSeconds = hour*secs_hour;
int32 minSeconds = min*secs_min;


//ADD 4 HOURS TO GET GMT UTC TIME FROM EST
int32 estToGMT = 4*secs_hour;
```

```
  int32 gmtUTC = yearSeconds +leapyearSeconds+ monthSeconds + daySeconds + hourSeconds +
minSeconds + estToGMT + 90062;

  return gmtUTC;

}
```

HAL_ACC.H

```
#ifndef HAL_ACC_H
#define HAL_ACC_H

#ifdef __cplusplus
extern "C" {
#endif

/*
-------------------------------------------------------------------------------------------
----
 *                                          Includes
 *
-------------------------------------------------------------------------------------------
----
 */

#include "comdef.h"

/*
-------------------------------------------------------------------------------------------
----
 *                                          Constants
 *
-------------------------------------------------------------------------------------------
----
 */
#define HAL_ACC_RANGE_8G       1
#define HAL_ACC_RANGE_4G       2
#define HAL_ACC_RANGE_2G       3

  /*
-------------------------------------------------------------------------------------------
----
 *                                           Typedefs
 *
-------------------------------------------------------------------------------------------
----
 */


/*
-------------------------------------------------------------------------------------------
----
 *                                          Functions
 *
-------------------------------------------------------------------------------------------
----
 */
```

```
void HalAccInit(void);
bool HalAccRead(uint8 *pBuf);
bool HalAccTest(void);
void HalAccSetRange(uint8 range);


/*********************************************************************************************
*******
*/

#ifdef __cplusplus
};
#endif

#endif

/*********************************************************************************************
*******
*/
```

HAL_ACC.C

```
/*
---------------------------------------------------------------------------------------------
----
*                                          Includes
*
---------------------------------------------------------------------------------------------
----
*/

#include "hal_acc.h"
#include "hal_sensor.h"
#include "hal_i2c.h"
#include "hal_board_cfg.h"

/*
---------------------------------------------------------------------------------------------
----
*                                           Constants
*
---------------------------------------------------------------------------------------------
----
*/
// Sensor I2C address
#define HAL_KXTI9_I2C_ADDRESS          0x0F

// KXTI9 register addresses
#define ACC_REG_ADDR_XOUT_HPF_L        0x00 // R
#define ACC_REG_ADDR_XOUT_HPF_H        0x01 // R
#define ACC_REG_ADDR_YOUT_HPF_L        0x02 // R
#define ACC_REG_ADDR_YOUT_HPF_H        0x03 // R
#define ACC_REG_ADDR_ZOUT_HPF_L        0x04 // R
#define ACC_REG_ADDR_ZOUT_HPF_H        0x05 // R
#define ACC_REG_ADDR_XOUT_L            0x06 // R
#define ACC_REG_ADDR_XOUT_H            0x07 // R
#define ACC_REG_ADDR_YOUT_L            0x08 // R
#define ACC_REG_ADDR_YOUT_H            0x09 // R
```

```
#define ACC_REG_ADDR_ZOUT_L             0x0A // R
#define ACC_REG_ADDR_ZOUT_H             0x0B // R
#define ACC_REG_ADDR_DCST_RESP          0x0C // R
#define ACC_REG_ADDR_WHO_AM_I           0x0F // R
#define ACC_REG_ADDR_TILT_POS_CUR       0x10 // R
#define ACC_REG_ADDR_TILT_POS_PRE       0x11 // R

#define ACC_REG_ADDR_INT_SRC_REG1       0x15 // R
#define ACC_REG_ADDR_INT_SRC_REG2       0x16 // R
#define ACC_REG_ADDR_STATUS_REG         0x18 // R
#define ACC_REG_ADDR_INT_REL            0x1A // R

#define ACC_REG_ADDR_CTRL_REG1          0x1B // R/W
#define ACC_REG_ADDR_CTRL_REG2          0x1C // R/W
#define ACC_REG_ADDR_CTRL_REG3          0x1D // R/W

#define ACC_REG_ADDR_INT_CTRL_REG1      0x1E // R/W
#define ACC_REG_ADDR_INT_CTRL_REG2      0x1F // R/W
#define ACC_REG_ADDR_INT_CTRL_REG3      0x20 // R/W
#define ACC_REG_ADDR_DATA_CTRL_REG      0x21 // R/W

#define ACC_REG_ADDR_TILT_TIMER         0x28 // R/W
#define ACC_REG_ADDR_WUF_TIMER          0x29 // R/W
#define ACC_REG_ADDR_TDT_TIMER          0x2B // R/W
#define ACC_REG_ADDR_TDT_H_THRESH       0x2C // R/W
#define ACC_REG_ADDR_TDT_L_THRESH       0x2D // R/W
#define ACC_REG_ADDR_TDT_TAP_TIMER      0x2E // R/W
#define ACC_REG_ADDR_TDT_TOTAL_TIMER    0x2F // R/W
#define ACC_REG_ADDR_TDT_LATENCY_TIMER  0x30 // R/W
#define ACC_REG_ADDR_TDT_WINDOW_TIMER   0x31 // R/W

#define ACC_REG_ADDR_BUF_CTRL1          0x32 // R/W
#define ACC_REG_ADDR_BUF_CTRL2          0x33 // R/W
#define ACC_REG_ADDR_BUF_STATUS_REG1    0x34 // R
#define ACC_REG_ADDR_BUF_STATUS_REG2    0x35 // R/W
#define ACC_REG_ADDR_BUF_CLEAR          0x36 // W

#define ACC_REG_ADDR_SELF_TEST          0x3A // R/W

#define ACC_REG_ADDR_WUF_THRESH         0x5A // R/W
#define ACC_REG_ADDR_TILT_ANGLE         0x5C // R/W
#define ACC_REG_ADDR_HYST_SET           0x5F // R/W
#define ACC_REG_ADDR_BUF_READ           0x7F // R/W

// Select register valies
#define REG_VAL_WHO_AM_I                0x08 // (data sheet says 0x04)

// CTRL1 BIT MASKS
#define ACC_REG_CTRL_PC                 0x80 // Power control  '1' On    '0' Off
#define ACC_REG_CTRL_RES                0x40 // Resolution     '1' High  '0' Low
#define ACC_REG_CTRL_DRDYE              0x20 // Data Ready     '1' On    '0' Off
#define ACC_REG_CTRL_GSEL_HI            0x10 // Range     '00' +/-2g    '01' +/-4g
#define ACC_REG_CTRL_GSEL_LO            0x08 //           '10' +/-8g    '11' N/A
#define ACC_REG_CTRL_GSEL_TDTE          0x04 // Directional Tap '1' On   '0' Off
#define ACC_REG_CTRL_GSEL_WUFE          0x02 // Wake Up         '1' On   '0' Off
#define ACC_REG_CTRL_GSEL_TPE           0x01 // Tilt Position   '1' On   '0' Off

// Range +- 2G
#define ACC_REG_CTRL_ON_2G              ( ACC_REG_CTRL_PC )
#define ACC_REG_CTRL_OFF_2G             ( 0 )
```

```
// Range +- 4G
#define ACC_REG_CTRL_ON_4G            ( ACC_REG_CTRL_PC | ACC_REG_CTRL_GSEL_LO)
#define ACC_REG_CTRL_OFF_4G           ( ACC_REG_CTRL_GSEL_LO )

// Range +- 8G
#define ACC_REG_CTRL_ON_8G            ( ACC_REG_CTRL_PC | ACC_REG_CTRL_GSEL_HI)
#define ACC_REG_CTRL_OFF_8G           ( ACC_REG_CTRL_GSEL_HI)



/*=========================================================================
ADXL345 SPECIFIC
 -------------------------------------------------------------------------*/
#define SENSORS_GRAVITY_EARTH             (9.80665F)              /**< Earth's gravity in
m/s^2 */
#define SENSORS_GRAVITY_MOON              (1.6F)                  /**< The moon's gravity in
m/s^2 */
#define SENSORS_GRAVITY_SUN               (275.0F)                /**< The sun's gravity in
m/s^2 */
#define SENSORS_GRAVITY_STANDARD          (SENSORS_GRAVITY_EARTH)

/*=========================================================================
 I2C ADDRESS/BITS
 -------------------------------------------------------------------------*/
#define ADXL345_ADDRESS                 (0x53)    // Assumes ALT address pin low
/*=========================================================================*/

/*=========================================================================
 REGISTERS
 -------------------------------------------------------------------------*/
#define ADXL345_REG_DEVID               (0x00)    // Device ID
#define ADXL345_REG_THRESH_TAP          (0x1D)    // Tap threshold
#define ADXL345_REG_OFSX                (0x1E)    // X-axis offset
#define ADXL345_REG_OFSY                (0x1F)    // Y-axis offset
#define ADXL345_REG_OFSZ                (0x20)    // Z-axis offset
#define ADXL345_REG_DUR                 (0x21)    // Tap duration
#define ADXL345_REG_LATENT              (0x22)    // Tap latency
#define ADXL345_REG_WINDOW              (0x23)    // Tap window
#define ADXL345_REG_THRESH_ACT          (0x24)    // Activity threshold
#define ADXL345_REG_THRESH_INACT        (0x25)    // Inactivity threshold
#define ADXL345_REG_TIME_INACT          (0x26)    // Inactivity time
#define ADXL345_REG_ACT_INACT_CTL       (0x27)    // Axis enable control for activity and
inactivity detection
#define ADXL345_REG_THRESH_FF           (0x28)    // Free-fall threshold
#define ADXL345_REG_TIME_FF             (0x29)    // Free-fall time
#define ADXL345_REG_TAP_AXES            (0x2A)    // Axis control for single/double tap
#define ADXL345_REG_ACT_TAP_STATUS      (0x2B)    // Source for single/double tap
#define ADXL345_REG_BW_RATE             (0x2C)    // Data rate and power mode control
#define ADXL345_REG_POWER_CTL           (0x2D)    // Power-saving features control
#define ADXL345_REG_INT_ENABLE          (0x2E)    // Interrupt enable control
#define ADXL345_REG_INT_MAP             (0x2F)    // Interrupt mapping control
#define ADXL345_REG_INT_SOURCE          (0x30)    // Source of interrupts
#define ADXL345_REG_DATA_FORMAT         (0x31)    // Data format control
#define ADXL345_REG_DATAX0              (0x32)    // X-axis data 0
#define ADXL345_REG_DATAX1              (0x33)    // X-axis data 1
#define ADXL345_REG_DATAY0              (0x34)    // Y-axis data 0
#define ADXL345_REG_DATAY1              (0x35)    // Y-axis data 1
#define ADXL345_REG_DATAZ0              (0x36)    // Z-axis data 0
#define ADXL345_REG_DATAZ1              (0x37)    // Z-axis data 1
```

```
#define ADXL345_REG_FIFO_CTL            (0x38)    // FIFO control
#define ADXL345_REG_FIFO_STATUS         (0x39)    // FIFO status
/*=========================================================================*/

/*=========================================================================
 REGISTERS
 -------------------------------------------------------------------------*/
#define ADXL345_MG2G_MULTIPLIER (0.004)  // 4mg per lsb
/*=========================================================================*/

/* Used with register 0x2C (ADXL345_REG_BW_RATE) to set bandwidth */
typedef enum {
       ADXL345_DATARATE_3200_HZ = 0x0F, // 1600Hz Bandwidth   140µA IDD
       ADXL345_DATARATE_1600_HZ = 0x0E, //  800Hz Bandwidth    90µA IDD
       ADXL345_DATARATE_800_HZ = 0x0D, //  400Hz Bandwidth   140µA IDD
       ADXL345_DATARATE_400_HZ = 0x0C, //  200Hz Bandwidth   140µA IDD
       ADXL345_DATARATE_200_HZ = 0x0B, //  100Hz Bandwidth   140µA IDD
       ADXL345_DATARATE_100_HZ = 0x0A, //   50Hz Bandwidth   140µA IDD
       ADXL345_DATARATE_50_HZ = 0x09, //   25Hz Bandwidth    90µA IDD
       ADXL345_DATARATE_25_HZ = 0x08, // 12.5Hz Bandwidth    60µA IDD
       ADXL345_DATARATE_12_5_HZ = 0x07, // 6.25Hz Bandwidth    50µA IDD
       ADXL345_DATARATE_6_25HZ = 0x06, // 3.13Hz Bandwidth    45µA IDD
       ADXL345_DATARATE_3_13_HZ = 0x05, // 1.56Hz Bandwidth    40µA IDD
       ADXL345_DATARATE_1_56_HZ = 0x04, // 0.78Hz Bandwidth    34µA IDD
       ADXL345_DATARATE_0_78_HZ = 0x03, // 0.39Hz Bandwidth    23µA IDD
       ADXL345_DATARATE_0_39_HZ = 0x02, // 0.20Hz Bandwidth    23µA IDD
       ADXL345_DATARATE_0_20_HZ = 0x01, // 0.10Hz Bandwidth    23µA IDD
       ADXL345_DATARATE_0_10_HZ = 0x00 // 0.05Hz Bandwidth    23µA IDD (default value)
} dataRate_t;

/* Used with register 0x31 (ADXL345_REG_DATA_FORMAT) to set g range */
typedef enum {
       ADXL345_RANGE_16_G = 0x03,   // +/- 16g
       ADXL345_RANGE_8_G = 0x02,   // +/- 8g
       ADXL345_RANGE_4_G = 0x01,   // +/- 4g
       ADXL345_RANGE_2_G = 0x00    // +/- 2g (default value)
} range_t;



/*
-------------------------------------------------------------------------------------------------
----
*                                          Typedefs
*
-------------------------------------------------------------------------------------------------
----
*/

/*
-------------------------------------------------------------------------------------------------
----
*                                          Macros
*
-------------------------------------------------------------------------------------------------
----
*/

/*
-------------------------------------------------------------------------------------------------
```

```
----
*                                          Local Functions
*
-------------------------------------------------------------------------------------------
----
*/
static void HalAccSelect(void);

/*
-------------------------------------------------------------------------------------------
----
*                                          Local Variables
*
-------------------------------------------------------------------------------------------
----
*/
static uint8 accSensorConfig;
static uint8 accSensorOff;
static uint8 accRange;

/*************************************************************************************
*******
* @fn          HalAccInit
*
* @brief       This function initializes the HAL Accelerometer abstraction layer.
*
* @return      None.
*/
void HalAccInit(void)
{
  HalAccSetRange(HAL_ACC_RANGE_8G);
}

/*************************************************************************************
*******
* @fn          HalAccSetRange
*
* @brief       Set the range of the accelerometer
*
* @param       range: HAL_ACC_RANGE_2G, HAL_ACC_RANGE_4G, HAL_ACC_RANGE_8G
*
* @return      None
*/
void HalAccSetRange(uint8 range)
{
  accRange = range;

  switch (accRange)
  {
  case HAL_ACC_RANGE_2G:
    accSensorConfig = ADXL345_RANGE_16_G;
    accSensorOff = ACC_REG_CTRL_OFF_2G;
    break;
  case HAL_ACC_RANGE_4G:
    accSensorConfig = ACC_REG_CTRL_ON_4G;
    accSensorOff = ACC_REG_CTRL_OFF_4G;
    break;
  case HAL_ACC_RANGE_8G:
    accSensorConfig = ACC_REG_CTRL_ON_8G;
    accSensorOff = ACC_REG_CTRL_OFF_8G;
```

```
    break;
  default:
    // Should not get here
    break;
  }
}

range_t adxl_range = ADXL345_RANGE_2_G;

void setRange(range_t range) {
       /* Red the data format register to preserve bits */

         uint8 registerPointer;

        HalSensorReadReg(ADXL345_REG_DATA_FORMAT,(uint8 *)&registerPointer, 1);

       uint8 format = registerPointer;

       /* Update the data rate */
       format &= ~0x0F;
       format |= range;

       /* Make sure that the FULL-RES bit is enabled for range scaling */
       format |= 0x08;

       /* Write the register back to the IC */

         uint8 value_pointer;

        value_pointer = format;

       HalSensorWriteReg(ADXL345_REG_DATA_FORMAT, &value_pointer,1);


       /* Keep track of the current range (to avoid readbacks) */
       adxl_range = range;
}

bool readAcc(uint8 *pBuf, int32 UTCTime){

  HalAccSelect();

  setRange(ADXL345_RANGE_16_G);

  uint8 registerPointer;

  bool success = HalSensorReadReg(ADXL345_REG_DEVID, (uint8 *)&registerPointer, 1);

       // printf("\nREGISTERPOINTER:%x\n", registerPointer);

       if(success){
         if (registerPointer == 0xe5){

            // Turn on sensor
  uint8 value_pointer = 0x08;

  HalSensorWriteReg(ADXL345_REG_POWER_CTL, &value_pointer, 1);

        }
```

```
      }

  //printf("INSIDE READACC\n");
  uint16 x;
  uint16 y;
  uint16 z;

  // Select this sensor




  // Wait for measurement ready (appx. 1.45 ms)
  ST_HAL_DELAY(180);

  // Read the three registers
  success = HalSensorReadReg(ADXL345_REG_DATAX0,(uint8 *)&x, 2);
  //success = HalSensorReadReg( ACC_REG_ADDR_XOUT_H, &x, sizeof(x));
  if (success)
  {
    success = HalSensorReadReg(ADXL345_REG_DATAY0,(uint8 *)&y, 2);
    if (success)
    {
       success = HalSensorReadReg(ADXL345_REG_DATAZ0,(uint8 *)&z, 2);
    }
  }

  if (success)
  {


  /*  printf("X: %u\n", x);
    printf("X: %x\n", x);


    printf("XACCL: %f\n", (float)((int)x * ADXL345_MG2G_MULTIPLIER *
SENSORS_GRAVITY_STANDARD));*/




    // Valid data

uint8 LSB_byte2 = (UTCTime & 0x000000ff);
uint8 LSB_byte1 = (UTCTime & 0x0000ff00) >> 8;
uint8 MSB_byte2 = (UTCTime & 0x00ff0000) >> 16;
uint8 MSB_byte1 = (UTCTime & 0xff000000) >> 24;


    pBuf[0] = 1;
    pBuf[1] = MSB_byte1;
    pBuf[2] = MSB_byte2;
    pBuf[3] = LSB_byte1;
    pBuf[4] = LSB_byte2;
    pBuf[5] = HI_UINT16( x );
    pBuf[6] = LO_UINT16( x );
    pBuf[7] = HI_UINT16( y );
    pBuf[8] = LO_UINT16( y );
    pBuf[9] = HI_UINT16( z );
```

```
    pBuf[10] = LO_UINT16( z );

   /* printf("HIGH: %u\n", HI_UINT16( x ));
     printf("HIGH: %x\n", HI_UINT16( x ));

    printf("LOW: %u\n", LO_UINT16( x ));
    printf("LOW: %x\n",LO_UINT16( x ));


    printf("HIGH+LOW: %u\n", BUILD_UINT16(LO_UINT16( x ),HI_UINT16( x )));
    printf("HIGH+LOW: %x\n", BUILD_UINT16(LO_UINT16( x ),HI_UINT16( x )));*/


  }

  // Turn off sensor


  // HalSensorWriteReg(ADXL345_REG_POWER_CTL, &value_pointer, 1);
    //HalSensorWriteReg(ADXL345_REG_POWER_CTL, &accSensorOff, sizeof(accSensorOff));

  return success;

}

/*************************************************************************************
*******
* @fn          HalAccRead
*
* @brief       Read data from the accelerometer - X, Y, Z - 3 bytes
*
* @return      TRUE if valid data, FALSE if not
*/
bool HalAccRead(uint8 *pBuf )
{
  uint8 x;
  uint8 y;
  uint8 z;
  bool success;

  // Select this sensor
  HalAccSelect();

  // Turn on sensor
  HalSensorWriteReg(ACC_REG_ADDR_CTRL_REG1, &accSensorConfig, sizeof(accSensorConfig));

  // Wait for measurement ready (appx. 1.45 ms)
  ST_HAL_DELAY(180);

  // Read the three registers
  success = HalSensorReadReg( ACC_REG_ADDR_XOUT_H, &x, sizeof(x));
  if (success)
  {
    success = HalSensorReadReg( ACC_REG_ADDR_YOUT_H, &y, sizeof(y));
    if (success)
    {
       success = HalSensorReadReg( ACC_REG_ADDR_ZOUT_H, &z, sizeof(z));
    }
  }
```

```
  if (success)
  {
    // Valid data
    pBuf[0] = x;
    pBuf[1] = y;
    pBuf[2] = z;
  }

  // Turn off sensor
  HalSensorWriteReg(ACC_REG_ADDR_CTRL_REG1, &accSensorOff, sizeof(accSensorOff));

  return success;
}


/*************************************************************************************************
 * @fn          HalAccTest
 *
 * @brief       Run a sensor self-test
 *
 * @return      TRUE if passed, FALSE if failed
 */
bool HalAccTest(void)
{
  uint8 val;

  // Select this sensor on the I2C bus
  HalAccSelect();

  // Check the DCST_RESP (pattern 0x55)
  ST_ASSERT(HalSensorReadReg(ACC_REG_ADDR_DCST_RESP, &val, 1));
  ST_ASSERT(val==0x55);

  // Check the DCST_RESP (pattern 0xAA)
  val = 0x10;     // Sets the DCST bit
  ST_ASSERT(HalSensorWriteReg(ACC_REG_ADDR_CTRL_REG3, &val, 1));
  ST_ASSERT(HalSensorReadReg(ACC_REG_ADDR_DCST_RESP, &val, 1));
  ST_ASSERT(val==0xAA);

  // Check the WHO AM I register
  ST_ASSERT(HalSensorReadReg(ACC_REG_ADDR_WHO_AM_I, &val, 1));
  ST_ASSERT(val==REG_VAL_WHO_AM_I);

  return TRUE;
}

/*
-------------------------------------------------------------------------------------------
----
*                                          Private functions
*
-------------------------------------------------------------------------------------------
-----
*/

/*************************************************************************************************
* @fn          HalAccSelect
*
```

```
* @brief       Select the accelerometer on the I2C-bus
*
* @return
*/
static void HalAccSelect(void)
{
  //Set up I2C that is used to communicate with SHT21
  HalI2CInit(ADXL345_ADDRESS ,i2cClock_267KHZ);
}

/*  Conversion algorithm for X, Y, Z
 *  ================================
 *
float calcAccel(int8 rawX, uint8 range)
{
    float v;

    switch (range)
    {
      case HAL_ACC_RANGE_2G:
      //-- calculate acceleration, unit G, range -2, +2
      v = (rawX * 1.0) / (256/4);
      break;

      case HAL_ACC_RANGE_4G:
      //-- calculate acceleration, unit G, range -4, +4
      v = (rawX * 1.0) / (256/8);
      break;

      case HAL_ACC_RANGE_4G:
      //-- calculate acceleration, unit G, range -8, +8
      v = (rawX * 1.0) / (256/16);
      break;
    }
    return v;
}
*/

/*********************************************************************
*********************************************************************/
```

HAL_IRTEMP.H

```
#ifndef HAL_IRTEMP_H
#define HAL_IRTEMP_H

#ifdef __cplusplus
extern "C"
{
#endif

/*********************************************************************
 * INCLUDES
 */
#include "comdef.h"

/*********************************************************************
```

```
 * CONSTANTS
 */


/*********************************************************************
 * TYPEDEFS
 */
typedef enum
{
  TMP006_OFF,               // IR Temperature Sleeping
  TMP006_IDLE,                      // IR Temperature On and Configured
  TMP006_DATA_READY         // IR Temperature On, Configured and Data is Ready
} IRTemperature_States_t;


/*********************************************************************
 * FUNCTIONS
 */
void HALIRTempInit(void);
void HalIRTempTurnOn(void);
void HalIRTempTurnOff(void);
bool HalIRTempRead(uint8 *irTempData, int32 UTCTime);
bool HalIRTempTest(void);
IRTemperature_States_t HalIRTempStatus(void);


#ifdef __cplusplus
}
#endif

#endif /* HAL_IRTEMP_H */
```

HAL_IRTEMP.C

```
/*
-------------------------------------------------------------------------------------------
----
*                                          Includes
*
-------------------------------------------------------------------------------------------
----
*/
#include "hal_irtemp.h"
#include "hal_i2c.h"
#include "hal_sensor.h"
#include "stdio.h"

/*
-------------------------------------------------------------------------------------------
----
*                                           Constants
*
-------------------------------------------------------------------------------------------
----
*/

/* Slave address */
```

```
#define TMP006_I2C_ADDRESS              0x40

/* TMP006 register addresses */
#define TMP006_REG_ADDR_VOLTAGE         0x00
#define TMP006_REG_ADDR_TEMPERATURE     0x01
#define TMP006_REG_ADDR_CONFIG          0x02
#define TMP006_REG_MANF_ID              0xFE
#define TMP006_REG_PROD_ID              0xFE

/* TMP006 register values */
#define TMP006_VAL_CONFIG_RESET         0x7400  // Sensor reset state
#define TMP006_VAL_CONFIG_ON            0x7000  // Sensor on state
#define TMP006_VAL_CONFIG_OFF           0x0000  // Sensor off state
#define TMP006_VAL_MANF_ID              0x5449  // Manufacturer ID
#define TMP006_VAL_PROD_ID              0x0067  // Product ID

/* Bit values */
#define DATA_RDY_BIT                    0x8000 // Data ready

/* Register length */
#define IRTEMP_REG_LEN                  2

#define TMP006_B0 -0.0000294
#define TMP006_B1 -0.00000057
#define TMP006_B2 0.00000000463
#define TMP006_C2 13.4
#define TMP006_TREF 298.15
#define TMP006_A2 -0.00001678
#define TMP006_A1 0.00175
#define TMP006_S0 6.4


/*
------------------------------------------------------------------------------------------
----
*                                          Local Functions
*
------------------------------------------------------------------------------------------
----
*/
static void HalIRTempSelect(void);


/*
------------------------------------------------------------------------------------------
----
*                                          Local Variables
*
------------------------------------------------------------------------------------------
----
*/
static IRTemperature_States_t irtSensorState = TMP006_OFF;

static uint8 configSensorReset[2] = {0x80, 0x00};  // Sensor reset
static uint8 configSensorOff[2] = {0x00, 0x80};    // Sensor standby
static uint8 configSensorOn[2] =  {0x70, 0x00};    // Conversion time 0.25 sec

/*
------------------------------------------------------------------------------------------
----
```

```
*                                          Public functions
*
-------------------------------------------------------------------------------------------
-----
*/


/*************************************************************************************
*******
 * @fn          HALIRTempInit
 *
 * @brief       Initialise the temperature sensor driver
 *
 * @return      none

*************************************************************************************
******/
void HALIRTempInit(void)
{
  irtSensorState = TMP006_OFF;
  HalIRTempTurnOff();
}


/*************************************************************************************
*******
 * @fn          HalIRTempTurnOn
 *
 * @brief       Turn the sensor on
 *
 * @return      none

*************************************************************************************
******/
void HalIRTempTurnOn(void)
{
  HalDcDcControl(ST_IRTEMP,true);
  HalIRTempSelect();

  if (HalSensorWriteReg(TMP006_REG_ADDR_CONFIG, configSensorOn, IRTEMP_REG_LEN))
  {
    irtSensorState = TMP006_IDLE;
  }
}

/*************************************************************************************
*******
 * @fn          HalIRTempTurnOff
 *
 * @brief       Turn the sensor off
 *
 * @return      none

*************************************************************************************
******/
void HalIRTempTurnOff(void)
{
  HalIRTempSelect();

  if (HalSensorWriteReg(TMP006_REG_ADDR_CONFIG, configSensorOff, IRTEMP_REG_LEN))
```

```
  {
    irtSensorState = TMP006_OFF;
  }
  HalDcDcControl(ST_IRTEMP,false);
}

double sqrt1(double x){



        long double x0 = 350;

        long double x1=0;

        for(int i=0; i<15; i++){

                x1 = (0.5)*(x0+(x/x0));

                x0 = x1;

        }

        return x0;
}

/*************************************************************************************************
 * @fn          HalIRTempRead
 *
 * @brief       Read the sensor voltage and sensor temperature registers
 *
 * @param       Voltage and temperature in raw format (2 + 2 bytes)
 *
 * @return      TRUE if valid data

*************************************************************************************************/
bool HalIRTempRead(uint8 *pBuf, int32 UTCTime)
{



  uint16 v;
  uint16 t;
  bool success;


  if (irtSensorState != TMP006_DATA_READY)
  {
    return FALSE;
  }

  HalIRTempSelect();

  // Read the sensor registers
  success = HalSensorReadReg(TMP006_REG_ADDR_VOLTAGE, (uint8 *)&v,IRTEMP_REG_LEN );
  if (success)
  {
    success = HalSensorReadReg(TMP006_REG_ADDR_TEMPERATURE, (uint8 *)&t,IRTEMP_REG_LEN );
  }
```

```
  if (success)
  {
    // Store values
    uint8 LSB_byte2 = (UTCTime & 0x000000ff);
    uint8 LSB_byte1 = (UTCTime & 0x0000ff00) >> 8;
    uint8 MSB_byte2 = (UTCTime & 0x00ff0000) >> 16;
    uint8 MSB_byte1 = (UTCTime & 0xff000000) >> 24;




    pBuf[0] = 4;
    pBuf[1] = MSB_byte1;
    pBuf[2] = MSB_byte2;
    pBuf[3] = LSB_byte1;
    pBuf[4] = LSB_byte2;
    pBuf[5] = 0;
    pBuf[6] = 0;
    pBuf[7] = LO_UINT16( v );
    pBuf[8] = HI_UINT16( v );
    pBuf[9] = LO_UINT16( t );
    pBuf[10] = HI_UINT16( t );




  }



  // Turn off sensor
  if (HalSensorWriteReg(TMP006_REG_ADDR_CONFIG, configSensorOff, IRTEMP_REG_LEN))
  {
    irtSensorState = TMP006_OFF;
  }
  HalDcDcControl(ST_IRTEMP,false);

  return success;
}




/*************************************************************************************************
*******
 * @fn          HalIRTempStatus
 *
 * @brief       Read the state of the sensor
 *
 * @return      none

*************************************************************************************************
******/
IRTemperature_States_t HalIRTempStatus(void)
{
  if (irtSensorState != TMP006_OFF)
  {
    bool success;
    uint16 v;
```

```
    // Select this sensor on the I2C bus
    HalIRTempSelect();

    // Read the data ready bit
    success = HalSensorReadReg(TMP006_REG_ADDR_CONFIG, (uint8 *)&v,IRTEMP_REG_LEN );
    if ((v & DATA_RDY_BIT) && success)
    {
      irtSensorState = TMP006_DATA_READY;
    }
  }

  return irtSensorState;
}


/*********************************************************************************************
*******
 * @fn          HalIRTempTest
 *
 * @brief       Run a sensor self-test
 *
 * @return      TRUE if passed, FALSE if failed

*********************************************************************************************
******/
bool HalIRTempTest(void)
{
  uint16 val;

  // Select this sensor on the I2C bus
  HalIRTempSelect();

  // Check manufacturer ID
  ST_ASSERT(HalSensorReadReg(TMP006_REG_MANF_ID, (uint8 *)&val, IRTEMP_REG_LEN));
  val = (LO_UINT16(val) << 8) | HI_UINT16(val);
  ST_ASSERT(val == TMP006_VAL_MANF_ID);

  // Reset sensor
  ST_ASSERT(HalSensorWriteReg(TMP006_REG_ADDR_CONFIG, configSensorReset, IRTEMP_REG_LEN));

  // Check config register (reset)
  ST_ASSERT(HalSensorReadReg(TMP006_REG_ADDR_CONFIG, (uint8 *)&val, IRTEMP_REG_LEN));
  val = ((LO_UINT16(val) << 8) | HI_UINT16(val));
  ST_ASSERT(val == TMP006_VAL_CONFIG_RESET);

  // Turn sensor off
  ST_ASSERT(HalSensorWriteReg(TMP006_REG_ADDR_CONFIG, configSensorOff,IRTEMP_REG_LEN));

  // Check config register (off)
  ST_ASSERT(HalSensorReadReg(TMP006_REG_ADDR_CONFIG, (uint8 *)&val, IRTEMP_REG_LEN));
  val = ((LO_UINT16(val) << 8) | HI_UINT16(val));
  ST_ASSERT(val == TMP006_VAL_CONFIG_OFF);

  // Turn sensor on
  ST_ASSERT(HalSensorWriteReg(TMP006_REG_ADDR_CONFIG, configSensorOn, IRTEMP_REG_LEN));

  // Check config register (on)
  ST_ASSERT(HalSensorReadReg(TMP006_REG_ADDR_CONFIG, (uint8 *)&val, IRTEMP_REG_LEN));
  val = ((LO_UINT16(val) << 8) | HI_UINT16(val));
```

```
  ST_ASSERT(val == TMP006_VAL_CONFIG_ON);

  // Turn sensor off
  ST_ASSERT(HalSensorWriteReg(TMP006_REG_ADDR_CONFIG, configSensorOff, IRTEMP_REG_LEN));

  return TRUE;
}

/*
-------------------------------------------------------------------------------------------
----
*                                          Private functions
*
-------------------------------------------------------------------------------------------
-----
*/

/*********************************************************************************************
*******
 * @fn          HalIRTempSelect
 *
 * @brief       Select the TMP006 slave and set the I2C bus speed
 *
 * @return      none

*********************************************************************************************
******/
static void HalIRTempSelect(void)
{
  // Select slave and set clock rate
  HalI2CInit(TMP006_I2C_ADDRESS, i2cClock_533KHZ);
}



/*  Conversion algorithm for die temperature
 *  ================================================
 *
double calcTmpLocal(uint16 rawT)
{
    //-- calculate die temperature [°C] --
    m_tmpAmb = (double)((qint16)rawT)/128.0;

    return m_tmpAmb;
}

*
* Conversion algorithm for target temperature
*
double calcTmpTarget(uint16 rawT)
{
    //-- calculate target temperature [°C] -
    double Vobj2 = (double)(qint16)rawT;
    Vobj2 *= 0.00000015625;

    double Tdie2 = m_tmpAmb + 273.15;
    const double S0 = 6.4E-14;            // Calibration factor

    const double a1 = 1.75E-3;
    const double a2 = -1.678E-5;
```

```
    const double b0 = -2.94E-5;
    const double b1 = -5.7E-7;
    const double b2 = 4.63E-9;
    const double c2 = 13.4;
    const double Tref = 298.15;
    double S = S0*(1+a1*(Tdie2 - Tref)+a2*pow((Tdie2 - Tref),2));
    double Vos = b0 + b1*(Tdie2 - Tref) + b2*pow((Tdie2 - Tref),2);
    double fObj = (Vobj2 - Vos) + c2*pow((Vobj2 - Vos),2);
    double tObj = pow(pow(Tdie2,4) + (fObj/S),.25);
    tObj = (tObj - 273.15);

    return tObj;
}

*/

/**********************************************************************
**********************************************************************/
```

Appendix K - Android Application Source Code

ADXL_GRAPHER.java

```
package com.example.android.bluetoothlegatt;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.TimeZone;

import com.jjoe64.graphview.CustomLabelFormatter;
import com.jjoe64.graphview.GraphView;
import com.jjoe64.graphview.GraphView.GraphViewData;
import com.jjoe64.graphview.GraphViewSeries;
import com.jjoe64.graphview.LineGraphView;

import android.app.Activity;
import android.os.Bundle;
import android.os.Environment;
import android.util.Log;
import android.widget.LinearLayout;

public class ADXL_GRAPHER extends Activity {

	private final static String TAG = ADXL_GRAPHER.class.getSimpleName();

	// String file = (String)
	// getIntent().getSerializableExtra("EXTRAS_FILE_NAME");
	// String file = "/sdcard/abc.csv";
	public static String file;

	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.test);

		Bundle extras = ADXL_GRAPHER.this.getIntent().getExtras();

		ArrayList<GraphViewData> data = new ArrayList<GraphViewData>();

		file = extras.getString("EXTRAS_FILE_NAME");

		file = Environment.getExternalStorageDirectory() + "/" + file;
		// file = "/sdcard/" + file;

		BufferedReader reader = null;
		try {
			reader = new BufferedReader(new FileReader(file));
		} catch (FileNotFoundException e1) {
			// TODO Auto-generated catch block
			e1.printStackTrace();
```

```
}
try {
	String line;
	while ((line = reader.readLine()) != null) {
		String[] RowData = line.split(",");
		String type = RowData[0].replace("", ' ').trim();

		String date = RowData[1].replace("", ' ').trim();

		String magnitude = RowData[2].replace("", ' ').trim();

		String utc = RowData[3].replace("", ' ').trim();

		if (type.equals("Accelerometer")) {

			Log.d(TAG, "IN HERE " + type);
			Log.d(TAG, "IN HERE UTC" + (double) Long.parseLong(utc));
			Log.d(TAG, "IN HERE MAG " + Double.parseDouble(magnitude));

			data.add(new GraphViewData((double) Long.parseLong(utc),
					Double.parseDouble(magnitude)));

		}
	}
} catch (IOException ex) {
	// handle exception
} finally {
	try {
		reader.close();
	} catch (IOException e) {
		// handle exception
	}
}

GraphView graphView = new LineGraphView(this,
		"Motion, magnitude (m/s^2)");

graphView.setCustomLabelFormatter(new CustomLabelFormatter() {

	@Override
	public String formatLabel(double value, boolean isValueX) {
		if (isValueX) {
			return getDateCurrentTimeZone((long) value);
		}
		return null;
	}
});

GraphViewData[] input = new GraphViewData[data.size()];

for (int i = 0; i < input.length; i++) {

	input[i] = data.get(i);
}

// add data
graphView.addSeries(new GraphViewSeries(input));
// set view port, start=2, size=40
graphView.setViewPort(2, 40);
graphView.setScrollable(true);
```

```
                // optional - activate scaling / zooming
                graphView.setScalable(true);

                LinearLayout layout = (LinearLayout) findViewById(R.id.graph1);
                layout.addView(graphView);
        }

        private String getDateCurrentTimeZone(long timestamp) {
                try {
                        Calendar calendar = Calendar.getInstance();
                        TimeZone tz = TimeZone.getDefault();
                        calendar.setTimeInMillis(timestamp * 1000);
                        calendar.add(Calendar.MILLISECOND, 0);
                        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
                        Date currenTimeZone = (Date) calendar.getTime();
                        return sdf.format(currenTimeZone);
                } catch (Exception e) {
                }
                return "";
        }

        /*
         * int num = 150;
         *
         * double v=0; for (int i=0; i<num; i++) { v += 0.2;
         *
         * }
         */
}
```

ADXL_Obj.java

```
package com.example.android.bluetoothlegatt;

import java.io.Serializable;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.TimeZone;


public class ADXL_Obj implements Serializable{

        private double ADXL345_MG2G_MULTIPLIER =  0.004;
        private double SENSORS_GRAVITY_STANDARD = 9.80665;

        private String date;
        private String data;

        private double x;
        private double y;
        private double z;

        private Integer UTC;

        public ADXL_Obj(Integer UTC, String data){

                this.UTC = UTC;
```

```
            this.date = getDateCurrentTimeZone(UTC);
            this.data = data;

    }

    public double getX(){

            String delimit[] = data.split(":");

            String xString = delimit[1];

            Integer decVal = Integer.parseInt(xString, 16);

            if(delimit[0].equalsIgnoreCase(("FF"))) decVal*=-1;

            return decVal * ADXL345_MG2G_MULTIPLIER * SENSORS_GRAVITY_STANDARD;
    }

    public double getY(){

            String delimit[] = data.split(":");

            String xString = delimit[3];

            Integer decVal = Integer.parseInt(xString, 16);

            if(delimit[2].equalsIgnoreCase(("FF"))) decVal*=-1;

            return decVal * ADXL345_MG2G_MULTIPLIER * SENSORS_GRAVITY_STANDARD;
    }

    public double getZ(){

            String delimit[] = data.split(":");

            String xString = delimit[5];

            Integer decVal = Integer.parseInt(xString, 16);

            if(delimit[4].equalsIgnoreCase(("FF"))) decVal*=-1;

            return decVal * ADXL345_MG2G_MULTIPLIER * SENSORS_GRAVITY_STANDARD;
    }

    public double getMagnitude(){

            double x = getX();
            double y = getY();
            double z = getZ();

            return Math.sqrt(Math.pow(x, 2) + Math.pow(y, 2) + Math.pow(z, 2));
    }

    public Integer getUTC(){
            return UTC;
    }

    public String getDate() {
            return date;
    }
```

```
        public void setDate(String date) {
                this.date = date;
        }

        public String getData() {
                return data;
        }

        public void setData(String data) {
                this.data = data;
        }



        private String getDateCurrentTimeZone(long timestamp) {
    try{
       Calendar calendar = Calendar.getInstance();
       TimeZone tz = TimeZone.getDefault();
       calendar.setTimeInMillis(timestamp * 1000);
       calendar.add(Calendar.MILLISECOND, 0);
       SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
       Date currenTimeZone = (Date) calendar.getTime();
       return sdf.format(currenTimeZone);
    }catch (Exception e) {
    }
    return "";
  }
}
```

BluetoothLeService.java

```
/*
 * Copyright (C) 2013 The Android Open Source Project
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package com.example.android.bluetoothlegatt;

import android.app.Service;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothGatt;
import android.bluetooth.BluetoothGattCallback;
import android.bluetooth.BluetoothGattCharacteristic;
import android.bluetooth.BluetoothGattService;
import android.bluetooth.BluetoothManager;
```

```
import android.bluetooth.BluetoothProfile;
import android.content.Context;
import android.content.Intent;
import android.os.Binder;
import android.os.IBinder;
import android.util.Log;

import java.util.List;

/**
 * Service for managing connection and data communication with a GATT server hosted on a
 * given Bluetooth LE device.
 */
public class BluetoothLeService extends Service {
    private final static String TAG = BluetoothLeService.class.getSimpleName();

    private BluetoothManager mBluetoothManager;
    private BluetoothAdapter mBluetoothAdapter;
    private String mBluetoothDeviceAddress;
    private BluetoothGatt mBluetoothGatt;
    private int mConnectionState = STATE_DISCONNECTED;

    private static final int STATE_DISCONNECTED = 0;
    private static final int STATE_CONNECTING = 1;
    private static final int STATE_CONNECTED = 2;

    public final static String ACTION_GATT_CONNECTED =
            "com.example.bluetooth.le.ACTION_GATT_CONNECTED";
    public final static String ACTION_GATT_DISCONNECTED =
            "com.example.bluetooth.le.ACTION_GATT_DISCONNECTED";
    public final static String ACTION_GATT_SERVICES_DISCOVERED =
            "com.example.bluetooth.le.ACTION_GATT_SERVICES_DISCOVERED";
    public final static String ACTION_DATA_AVAILABLE =
            "com.example.bluetooth.le.ACTION_DATA_AVAILABLE";
    public final static String EXTRA_DATA =
            "com.example.bluetooth.le.EXTRA_DATA";


    // Implements callback methods for GATT events that the app cares about.  For example,
    // connection change and services discovered.
    private final BluetoothGattCallback mGattCallback = new BluetoothGattCallback() {
        @Override
        public void onConnectionStateChange(BluetoothGatt gatt, int status, int newState) {
            String intentAction;
            if (newState == BluetoothProfile.STATE_CONNECTED) {
              Log.i("STATE CONNECTED", "OK");
                intentAction = ACTION_GATT_CONNECTED;
                mConnectionState = STATE_CONNECTED;
                broadcastUpdate(intentAction);
                Log.i("STATE CONNECTED BROADCAST SENT - DISCOVERING SERVICES", "OK");
                // Attempts to discover services after successful connection.
                mBluetoothGatt.discoverServices();
                Log.i("AFTER DISCOVER SERVICES", "OK");

            } else if (newState == BluetoothProfile.STATE_DISCONNECTED) {
              Log.i("STATE DISCONNECTED", "OK");
              intentAction = ACTION_GATT_DISCONNECTED;
                mConnectionState = STATE_DISCONNECTED;
                Log.i(TAG, "Disconnected from GATT server.");
                broadcastUpdate(intentAction);
```

```
            Log.i("STATE DISCONNECTED BROADCAST SENT", "OK");
        }

        if(mConnectionState==123334){

        }
    }

    @Override
    public void onServicesDiscovered(BluetoothGatt gatt, int status) {
        if (status == BluetoothGatt.GATT_SUCCESS) {
            broadcastUpdate(ACTION_GATT_SERVICES_DISCOVERED);
            Log.i("GATT SUCCESS - SERVICES DISCOVERED", "OK");
        } else {
            Log.w(TAG, "onServicesDiscovered received: " + status);
        }
    }

    @Override
    public void onCharacteristicRead(BluetoothGatt gatt,
                                     BluetoothGattCharacteristic characteristic,
                                     int status) {
        if (status == BluetoothGatt.GATT_SUCCESS) {
            broadcastUpdate(ACTION_DATA_AVAILABLE, characteristic);
        }
    }

    @Override
    public void onCharacteristicChanged(BluetoothGatt gatt,
                                        BluetoothGattCharacteristic characteristic) {
        broadcastUpdate(ACTION_DATA_AVAILABLE, characteristic);
    }
};

private void broadcastUpdate(final String action) {
    final Intent intent = new Intent(action);
    sendBroadcast(intent);
    Log.i("INTENT SENT", "OK");
}

private void broadcastUpdate(final String action, final BluetoothGattCharacteristic characteristic) {
    final Intent intent = new Intent(action);

        // For all other profiles, writes the data formatted in HEX.
        final byte[] data = characteristic.getValue();
        if (data != null && data.length > 0) {
            final StringBuilder stringBuilder = new StringBuilder(data.length);
            for(byte byteChar : data)
                stringBuilder.append(String.format("%02X ", byteChar));
            intent.putExtra(EXTRA_DATA, new String(data) + "\n" + stringBuilder.toString());
    }
    sendBroadcast(intent);
}

public class LocalBinder extends Binder {
    BluetoothLeService getService() {
        return BluetoothLeService.this;
    }
}
```

```
@Override
public IBinder onBind(Intent intent) {
    return mBinder;
}

@Override
public boolean onUnbind(Intent intent) {
    // After using a given device, you should make sure that BluetoothGatt.close() is called
    // such that resources are cleaned up properly.  In this particular example, close() is
    // invoked when the UI is disconnected from the Service.
    close();
    return super.onUnbind(intent);
}

private final IBinder mBinder = new LocalBinder();

/**
 * Initializes a reference to the local Bluetooth adapter.
 *
 * @return Return true if the initialization is successful.
 */
public boolean initialize() {
    // For API level 18 and above, get a reference to BluetoothAdapter through
    // BluetoothManager.
    if (mBluetoothManager == null) {
        mBluetoothManager = (BluetoothManager) getSystemService(Context.BLUETOOTH_SERVICE);
        if (mBluetoothManager == null) {
            Log.e(TAG, "Unable to initialize BluetoothManager.");
            return false;
        }
    }

    mBluetoothAdapter = mBluetoothManager.getAdapter();
    if (mBluetoothAdapter == null) {
        Log.e(TAG, "Unable to obtain a BluetoothAdapter.");
        return false;
    }

    return true;
}

/**
 * Connects to the GATT server hosted on the Bluetooth LE device.
 *
 * @param address The device address of the destination device.
 *
 * @return Return true if the connection is initiated successfully. The connection result
 *         is reported asynchronously through the
 *         {@code BluetoothGattCallback#onConnectionStateChange(android.bluetooth.BluetoothGatt, int, int)}
 *         callback.
 */
public boolean connect(final String address) {
    if (mBluetoothAdapter == null || address == null) {
        Log.w(TAG, "BluetoothAdapter not initialized or unspecified address.");
        return false;
    }

    // Previously connected device.  Try to reconnect.
    if (mBluetoothDeviceAddress != null && address.equals(mBluetoothDeviceAddress)
            && mBluetoothGatt != null) {
```

```
            Log.d(TAG, "Trying to use an existing mBluetoothGatt for connection.");
            if (mBluetoothGatt.connect()) {
                mConnectionState = STATE_CONNECTING;
                return true;
            } else {
                return false;
            }
        }

        final BluetoothDevice device = mBluetoothAdapter.getRemoteDevice(address);
        if (device == null) {
            Log.w(TAG, "Device not found.  Unable to connect.");
            return false;
        }
        // We want to directly connect to the device, so we are setting the autoConnect
        // parameter to false.
        mBluetoothGatt = device.connectGatt(this, false, mGattCallback);
        Log.d(TAG, "Trying to create a new connection.");
        mBluetoothDeviceAddress = address;
        mConnectionState = STATE_CONNECTING;
        return true;
    }

    /**
     * Disconnects an existing connection or cancel a pending connection. The disconnection result
     * is reported asynchronously through the
     * {@code BluetoothGattCallback#onConnectionStateChange(android.bluetooth.BluetoothGatt, int, int)}
     * callback.
     */
    public void disconnect() {
        if (mBluetoothAdapter == null || mBluetoothGatt == null) {
            Log.w(TAG, "BluetoothAdapter not initialized");
            return;
        }
        mBluetoothGatt.disconnect();
    }

    /**
     * After using a given BLE device, the app must call this method to ensure resources are
     * released properly.
     */
    public void close() {
        if (mBluetoothGatt == null) {
            return;
        }
        mBluetoothGatt.close();
        mBluetoothGatt = null;
    }

    /**
     * Request a read on a given {@code BluetoothGattCharacteristic}. The read result is reported
     * asynchronously through the {@code
BluetoothGattCallback#onCharacteristicRead(android.bluetooth.BluetoothGatt,
android.bluetooth.BluetoothGattCharacteristic, int)}
     * callback.
     *
     * @param characteristic The characteristic to read from.
     */
    public void readCharacteristic(BluetoothGattCharacteristic characteristic) {
        if (mBluetoothAdapter == null || mBluetoothGatt == null) {
```

```
            Log.w(TAG, "BluetoothAdapter not initialized");
            return;
        }
        mBluetoothGatt.readCharacteristic(characteristic);
    }

    /**
     * Enables or disables notification on a give characteristic.
     *
     * @param characteristic Characteristic to act on.
     * @param enabled If true, enable notification.  False otherwise.
     */
    public void setCharacteristicNotification(BluetoothGattCharacteristic characteristic,
                                              boolean enabled) {
        if (mBluetoothAdapter == null || mBluetoothGatt == null) {
            Log.w(TAG, "BluetoothAdapter not initialized");
            return;
        }
        mBluetoothGatt.setCharacteristicNotification(characteristic, enabled);

    }

    /**
     * Retrieves a list of supported GATT services on the connected device. This should be
     * invoked only after {@code BluetoothGatt#discoverServices()} completes successfully.
     *
     * @return A {@code List} of supported services.
     */
    public List<BluetoothGattService> getSupportedGattServices() {
        if (mBluetoothGatt == null) return null;

        return mBluetoothGatt.getServices();
    }

    public BluetoothGatt getGatt(){

            return mBluetoothGatt;
    }
}
```

BPM_GRAPHER.java

```
package com.example.android.bluetoothlegatt;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.TimeZone;

import com.jjoe64.graphview.CustomLabelFormatter;
import com.jjoe64.graphview.GraphView;
```

```
import com.jjoe64.graphview.GraphView.GraphViewData;
import com.jjoe64.graphview.GraphViewSeries;
import com.jjoe64.graphview.LineGraphView;

import android.app.Activity;
import android.os.Bundle;
import android.os.Environment;
import android.util.Log;
import android.widget.LinearLayout;

public class BPM_GRAPHER extends Activity {

	private final static String TAG = BPM_GRAPHER.class.getSimpleName();

	// String file = (String)
	// getIntent().getSerializableExtra("EXTRAS_FILE_NAME");
	// String file = "/sdcard/abc.csv";
	public static String file;

	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.test);

		Bundle extras = BPM_GRAPHER.this.getIntent().getExtras();

		ArrayList<GraphViewData> data = new ArrayList<GraphViewData>();

		file = extras.getString("EXTRAS_FILE_NAME");

		file = Environment.getExternalStorageDirectory() + "/" + file;
		// file = "/sdcard/" + file;

		BufferedReader reader = null;
		try {
			reader = new BufferedReader(new FileReader(file));
		} catch (FileNotFoundException e1) {
			// TODO Auto-generated catch block
			e1.printStackTrace();
		}
		try {
			String line;
			while ((line = reader.readLine()) != null) {
				String[] RowData = line.split(",");
				String type = RowData[0].replace("", ' ').trim();

				String date = RowData[1].replace("", ' ').trim();

				String bpm = RowData[2].replace("", ' ').trim();

				String utc = RowData[3].replace("", ' ').trim();

				if (type.equals("BPM")) {

					Log.d(TAG, "IN HERE " + type);
					Log.d(TAG, "IN HERE UTC" + (double) Long.parseLong(utc));
					Log.d(TAG, "IN HERE BPM " + Integer.parseInt(bpm));

					data.add(new GraphViewData((double) Long.parseLong(utc),
							Double.parseDouble(bpm)));
```

```
                                }
                        }
                } catch (IOException ex) {
                        // handle exception
                } finally {
                        try {
                                reader.close();
                        } catch (IOException e) {
                                // handle exception
                        }
                }

                GraphView graphView = new LineGraphView(this, "Beats Per Minute");

                graphView.setCustomLabelFormatter(new CustomLabelFormatter() {

                        @Override
                        public String formatLabel(double value, boolean isValueX) {
                                if (isValueX) {
                                        return getDateCurrentTimeZone((long) value);
                                }
                                return null;
                        }
                });

                GraphViewData[] input = new GraphViewData[data.size()];

                for (int i = 0; i < input.length; i++) {

                        input[i] = data.get(i);
                }

                // add data
                graphView.addSeries(new GraphViewSeries(input));
                // set view port, start=2, size=40
                graphView.setViewPort(2, 40);
                graphView.setScrollable(true);
                // optional - activate scaling / zooming
                graphView.setScalable(true);

                LinearLayout layout = (LinearLayout) findViewById(R.id.graph1);
                layout.addView(graphView);
        }

        private String getDateCurrentTimeZone(long timestamp) {
                try {
                        Calendar calendar = Calendar.getInstance();
                        TimeZone tz = TimeZone.getDefault();
                        calendar.setTimeInMillis(timestamp * 1000);
                        calendar.add(Calendar.MILLISECOND, 0);
                        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
                        Date currenTimeZone = (Date) calendar.getTime();
                        return sdf.format(currenTimeZone);
                } catch (Exception e) {
                }
                return "";
        }

        /*
         * int num = 150;
```

```
            *
            * double v=0; for (int i=0; i<num; i++) { v += 0.2;
            *
            * }
            */
}
```

BPM_Obj.java

```
package com.example.android.bluetoothlegatt;

import java.io.Serializable;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.TimeZone;

public class BPM_Obj implements Serializable{

            private String date;
            private String data;

            public BPM_Obj(Integer UTC, String data){

                        this.date = getDateCurrentTimeZone(UTC);
                        this.data = data;

            }



            private String getDateCurrentTimeZone(long timestamp) {
        try{
           Calendar calendar = Calendar.getInstance();
           TimeZone tz = TimeZone.getDefault();
           calendar.setTimeInMillis(timestamp * 1000);
           calendar.add(Calendar.MILLISECOND, 0);
           SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
           Date currenTimeZone = (Date) calendar.getTime();
           return sdf.format(currenTimeZone);
        }catch (Exception e) {
        }
        return "";
    }
}
```

CopyOfGrapher.java

```
package com.example.android.bluetoothlegatt;

import java.util.ArrayList;

import com.jjoe64.graphview.GraphView;
import com.jjoe64.graphview.GraphView.GraphViewData;
import com.jjoe64.graphview.GraphViewSeries;
import com.jjoe64.graphview.LineGraphView;
```

```
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.widget.LinearLayout;

public class CopyOfGrapher extends Activity{

   private ArrayList<ADXL_Obj> ADXL_DATA;
   //private ArrayList<LIGHT_Obj> LIGHT_DATA = (ArrayList<LIGHT_Obj>)
getIntent().getSerializableExtra("light_data");
   //private ArrayList<BPM_Obj> BPM_DATA = (ArrayList<BPM_Obj>)
getIntent().getSerializableExtra("bpm_data");
   //private ArrayList<TMP006_Obj> TMP_DATA = (ArrayList<TMP006_Obj>)
getIntent().getSerializableExtra("tmp006_data");

        private final static String TAG = CopyOfGrapher.class.getSimpleName();

        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.test);

                ADXL_DATA = (ArrayList<ADXL_Obj>) getIntent().getSerializableExtra("adxl_data");

        if(ADXL_DATA.size()>0){

                GraphViewData[] data = new GraphViewData[ADXL_DATA.size()];

                int i = 0;

                        for(ADXL_Obj adxl : ADXL_DATA){
                                Log.d(TAG, "DATA: " + adxl.getData());
                                Log.d(TAG, "DATAX: " + adxl.getX());
                                Log.d(TAG, "DATAY: " + adxl.getY());
                                Log.d(TAG, "DATAZ: " + adxl.getZ());
                                Log.d(TAG, "DATAMAG: " + adxl.getMagnitude());

                                 data[i] = new GraphViewData(adxl.getUTC(), (adxl.getMagnitude()));
                                 i++;
                        }

                        GraphView graphView = new LineGraphView(
                                   this
                                   , "GraphViewDemo"
                                );

                        // add data
                        graphView.addSeries(new GraphViewSeries(data));
                        // set view port, start=2, size=40
                        graphView.setViewPort(2, 40);
                        graphView.setScrollable(true);
                        // optional - activate scaling / zooming
                        graphView.setScalable(true);

                        LinearLayout layout = (LinearLayout) findViewById(R.id.graph1);
                        layout.addView(graphView);
                }

                /*int num = 150;

                double v=0;
```

```
                    for (int i=0; i<num; i++) {
                      v += 0.2;
                    }*/
            }
}
```

csv_data_selector.java

```
package com.example.android.bluetoothlegatt;

import java.io.File;
import java.util.ArrayList;

import com.jjoe64.graphview.GraphView;
import com.jjoe64.graphview.GraphView.GraphViewData;
import com.jjoe64.graphview.GraphViewSeries;
import com.jjoe64.graphview.LineGraphView;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.os.Environment;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ExpandableListView;
import android.widget.LinearLayout;
import android.widget.ListView;
import android.widget.TextView;

public class csv_data_selector extends Activity{

        public static String filename = null;
        public static String EXTRAS_FILE_NAME;
        private final static String TAG = csv_data_selector.class
                        .getSimpleName();



        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.csv_dataviews);

                Bundle extras = csv_data_selector.this.getIntent().getExtras();

                filename = extras.getString("EXTRAS_FILE_NAME");

        }

        public void onClick(View v){

           if(v.getId() == R.id.viewADXL){
                        Intent ADXLintent = new Intent(this, ADXL_GRAPHER.class);
                        ADXLintent.putExtra("EXTRAS_FILE_NAME", filename);
                    startActivity(ADXLintent);
```

```
            }else if(v.getId() == R.id.viewVISLIGHT){

                        Intent LIGHTintent = new Intent(this, VISLIGHT_GRAPHER.class);
                        LIGHTintent.putExtra("EXTRAS_FILE_NAME", filename);
                startActivity(LIGHTintent);

            }
            else if(v.getId() == R.id.viewUVLIGHT){

                        Intent LIGHTintent = new Intent(this, UVLIGHT_GRAPHER.class);
                        LIGHTintent.putExtra("EXTRAS_FILE_NAME", filename);
                startActivity(LIGHTintent);

            }
            else if(v.getId() == R.id.viewIRLIGHT){

                        Intent LIGHTintent = new Intent(this, IRLIGHT_GRAPHER.class);
                        LIGHTintent.putExtra("EXTRAS_FILE_NAME", filename);
                startActivity(LIGHTintent);

            }
            else if(v.getId() == R.id.viewTEMP){

                        Intent LIGHTintent = new Intent(this, TEMPERATURE_GRAPHER.class);
                        LIGHTintent.putExtra("EXTRAS_FILE_NAME", filename);
                startActivity(LIGHTintent);

            }
            else if(v.getId() == R.id.viewBPM){

                        Intent BPMintent = new Intent(this, BPM_GRAPHER.class);
                        BPMintent.putExtra("EXTRAS_FILE_NAME", filename);
                startActivity(BPMintent);
            }
        }
}
```

csv_select.java

```
package com.example.android.bluetoothlegatt;

import java.io.File;
import java.util.ArrayList;
import java.util.Collections;

import com.jjoe64.graphview.GraphView;
import com.jjoe64.graphview.GraphView.GraphViewData;
import com.jjoe64.graphview.GraphViewSeries;
import com.jjoe64.graphview.LineGraphView;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.os.Environment;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
```

```
import android.widget.Button;
import android.widget.ExpandableListView;
import android.widget.LinearLayout;
import android.widget.ListView;
import android.widget.TextView;

public class csv_select extends Activity{

	public static String filename = null;
	public static String EXTRAS_FILE_NAME;
	private final static String TAG = csv_select.class
				.getSimpleName();



	private ListView csv_list;

	public ArrayList<String> GetFiles(String DirectoryPath) {
	   ArrayList<String> MyFiles = new ArrayList<String>();
	   File f = new File(DirectoryPath);

	   f.mkdirs();
	   File[] files = f.listFiles();
	   if (files.length == 0)
	      return null;
	   else {
	      for (int i=0; i<files.length; i++)
	           if(files[i].toString().contains("csv")){
	         MyFiles.add(files[i].getName());
	           }
	   }

	   Collections.sort(MyFiles);
	   Collections.reverse(MyFiles);

	   return MyFiles;
	}

	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.csv_select);

		//csv_list = (ListView) findViewById(R.id.CSV_FILES_LIST);


    //mGattServicesList.setOnChildClickListener(servicesListClickListner);

	}

	public void onClick(View v){

	   if(v.getId() == R.id.SyncPhone){
		    Intent syncPhoneIntent = new Intent(this, DeviceScanActivity.class);
		       startActivity(syncPhoneIntent);

	   }else if(v.getId() == R.id.readCSV){

		TextView fileselect=(TextView)findViewById(R.id.selectafiletext);
		fileselect.setVisibility(0); //To set visible
```

```
                    ArrayList<String> FilesInFolder =
GetFiles(Environment.getExternalStorageDirectory().toString() + "/");
                        csv_list = (ListView)findViewById(R.id.CSV_FILES_LIST);

                        csv_list.setAdapter(new ArrayAdapter<String>(this,
                           android.R.layout.simple_list_item_1, FilesInFolder));

                        csv_list.setOnItemClickListener(new AdapterView.OnItemClickListener() {
                        public void onItemClick(AdapterView<?> parent, View v, int position, long id) {

                                filename = csv_list.getItemAtPosition(position).toString();

                                   Log.d(TAG, "TESTING " + filename);

                                        Intent graphintent = new Intent(getBaseContext(), csv_data_selector.class);
                                   graphintent.putExtra("EXTRAS_FILE_NAME", filename);

                                   startActivity(graphintent);

                           }
                        });
                }
            }
}
```

DeviceControlActivity.java

```
package com.example.android.bluetoothlegatt;

import android.app.Activity;
import android.bluetooth.BluetoothGattCharacteristic;
import android.bluetooth.BluetoothGattService;
import android.content.BroadcastReceiver;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.IBinder;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.ExpandableListView;
import android.widget.SimpleExpandableListAdapter;
import android.widget.TextView;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

/**
 * For a given BLE device, this Activity provides the user interface to connect, display data,
 * and display GATT services and characteristics supported by the device.  The Activity
 * communicates with {@code BluetoothLeService}, which in turn interacts with the
 * Bluetooth LE API.
 */
```

```
public class DeviceControlActivity extends Activity {
    private final static String TAG = DeviceControlActivity.class.getSimpleName();

    public static final String EXTRAS_DEVICE_NAME = "DEVICE_NAME";
    public static final String EXTRAS_DEVICE_ADDRESS = "DEVICE_ADDRESS";

    private TextView mConnectionState;
    private TextView mDataField;
    private String mDeviceName;
    private String mDeviceAddress;
    private ExpandableListView mGattServicesList;
    private BluetoothLeService mBluetoothLeService;
    private ArrayList<ArrayList<BluetoothGattCharacteristic>> mGattCharacteristics =
            new ArrayList<ArrayList<BluetoothGattCharacteristic>>();
    private boolean mConnected = false;
    private BluetoothGattCharacteristic mNotifyCharacteristic;

    private final String LIST_NAME = "NAME";
    private final String LIST_UUID = "UUID";

    // Code to manage Service lifecycle.
    private final ServiceConnection mServiceConnection = new ServiceConnection() {

        @Override
        public void onServiceConnected(ComponentName componentName, IBinder service) {
            mBluetoothLeService = ((BluetoothLeService.LocalBinder) service).getService();
            mBluetoothLeService.initialize();
            if (!mBluetoothLeService.initialize()) {
                Log.e(TAG, "Unable to initialize Bluetooth");
                finish();
            }
            // Automatically connects to the device upon successful start-up initialization.
            mBluetoothLeService.connect(mDeviceAddress);
        }

        @Override
        public void onServiceDisconnected(ComponentName componentName) {
            mBluetoothLeService = null;
        }
    };

    // Handles various events fired by the Service.
    // ACTION_GATT_CONNECTED: connected to a GATT server.
    // ACTION_GATT_DISCONNECTED: disconnected from a GATT server.
    // ACTION_GATT_SERVICES_DISCOVERED: discovered GATT services.
    // ACTION_DATA_AVAILABLE: received data from the device.  This can be a result of read
    //                        or notification operations.
    private final BroadcastReceiver mGattUpdateReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            final String action = intent.getAction();
            if (BluetoothLeService.ACTION_GATT_CONNECTED.equals(action)) {
              Log.i("ON GATT CONNECTED 1", "OK");
              mConnected = true;
                updateConnectionState(R.string.connected);
                invalidateOptionsMenu();
                Log.i("ON GATT CONNECTED 2", "OK");
            } else if (BluetoothLeService.ACTION_GATT_DISCONNECTED.equals(action)) {
                mConnected = false;
                updateConnectionState(R.string.disconnected);
```

```
            invalidateOptionsMenu();
            clearUI();
            Log.i("ON GATT DISCONNECTED", "OK");
        } else if (BluetoothLeService.ACTION_GATT_SERVICES_DISCOVERED.equals(action)) {
            // Show all the supported services and characteristics on the user interface.
          Log.i("ON GATT SERVICE DISCOVERED", "OK");
            displayGattServices(mBluetoothLeService.getSupportedGattServices());
            Log.i("ON DISPLAYED SERVICES", "OK");
        } else if (BluetoothLeService.ACTION_DATA_AVAILABLE.equals(action)) {
          Log.i("ON GATT DATA AVAILABLE", "OK");
            displayData(intent.getStringExtra(BluetoothLeService.EXTRA_DATA));
        }
    }
};

// If a given GATT characteristic is selected, check for supported features.  This sample
// demonstrates 'Read' and 'Notify' features.  See
// http://d.android.com/reference/android/bluetooth/BluetoothGatt.html for the complete
// list of supported characteristic features.
private final ExpandableListView.OnChildClickListener servicesListClickListner =
        new ExpandableListView.OnChildClickListener() {
            @Override
            public boolean onChildClick(ExpandableListView parent, View v, int groupPosition,
                                        int childPosition, long id) {
                if (mGattCharacteristics != null) {
                    final BluetoothGattCharacteristic characteristic =
                            mGattCharacteristics.get(groupPosition).get(childPosition);
                    final int charaProp = characteristic.getProperties();
                    if ((charaProp | BluetoothGattCharacteristic.PROPERTY_READ) > 0) {
                        // If there is an active notification on a characteristic, clear
                        // it first so it doesn't update the data field on the user interface.
                        if (mNotifyCharacteristic != null) {
                            mBluetoothLeService.setCharacteristicNotification(
                                    mNotifyCharacteristic, false);
                            mNotifyCharacteristic = null;
                        }
                        mBluetoothLeService.readCharacteristic(characteristic);
                    }
                    if ((charaProp | BluetoothGattCharacteristic.PROPERTY_NOTIFY) > 0) {
                        mNotifyCharacteristic = characteristic;
                        mBluetoothLeService.setCharacteristicNotification(
                                characteristic, true);
                    }
                    return true;
                }
                return false;
            }
};

private void clearUI() {
    mGattServicesList.setAdapter((SimpleExpandableListAdapter) null);
    mDataField.setText(R.string.no_data);
}

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.gatt_services_characteristics);

    final Intent intent = getIntent();
```

```
    mDeviceName = intent.getStringExtra(EXTRAS_DEVICE_NAME);
    mDeviceAddress = intent.getStringExtra(EXTRAS_DEVICE_ADDRESS);

    // Sets up UI references.
    ((TextView) findViewById(R.id.device_address)).setText(mDeviceAddress);
    mGattServicesList = (ExpandableListView) findViewById(R.id.gatt_services_list);
    mGattServicesList.setOnChildClickListener(servicesListClickListner);
    mConnectionState = (TextView) findViewById(R.id.connection_state);
    mDataField = (TextView) findViewById(R.id.data_value);

    getActionBar().setTitle(mDeviceName);
    getActionBar().setDisplayHomeAsUpEnabled(true);
    Intent gattServiceIntent = new Intent(this, BluetoothLeService.class);
    bindService(gattServiceIntent, mServiceConnection, BIND_AUTO_CREATE);
}

@Override
protected void onResume() {
    super.onResume();
    registerReceiver(mGattUpdateReceiver, makeGattUpdateIntentFilter());
    if (mBluetoothLeService != null) {
        final boolean result = mBluetoothLeService.connect(mDeviceAddress);
        Log.d(TAG, "Connect request result=" + result);
    }
}

@Override
protected void onPause() {
    super.onPause();
    unregisterReceiver(mGattUpdateReceiver);
}

@Override
protected void onDestroy() {
    super.onDestroy();
    unbindService(mServiceConnection);
    mBluetoothLeService = null;
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.gatt_services, menu);
    if (mConnected) {
        menu.findItem(R.id.menu_connect).setVisible(false);
        menu.findItem(R.id.menu_disconnect).setVisible(true);
    } else {
        menu.findItem(R.id.menu_connect).setVisible(true);
        menu.findItem(R.id.menu_disconnect).setVisible(false);
    }
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch(item.getItemId()) {
        case R.id.menu_connect:
            mBluetoothLeService.connect(mDeviceAddress);
            return true;
        case R.id.menu_disconnect:
            mBluetoothLeService.disconnect();
```

```
            return true;
        case android.R.id.home:
            onBackPressed();
            return true;
    }
    return super.onOptionsItemSelected(item);
}

private void updateConnectionState(final int resourceId) {
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            mConnectionState.setText(resourceId);
        }
    });
}

private void displayData(String data) {
    if (data != null) {
        mDataField.setText(data);
    }
}

// Demonstrates how to iterate through the supported GATT Services/Characteristics.
// In this sample, we populate the data structure that is bound to the ExpandableListView
// on the UI.
private void displayGattServices(List<BluetoothGattService> gattServices) {
    if (gattServices == null) return;
    String uuid = null;
    ArrayList<HashMap<String, String>> gattServiceData = new ArrayList<HashMap<String, String>>();
    ArrayList<ArrayList<HashMap<String, String>>> gattCharacteristicData
            = new ArrayList<ArrayList<HashMap<String, String>>>();
    mGattCharacteristics = new ArrayList<ArrayList<BluetoothGattCharacteristic>>();

    // Loops through available GATT Services.
    for (BluetoothGattService gattService : gattServices) {
        HashMap<String, String> currentServiceData = new HashMap<String, String>();
        uuid = gattService.getUuid().toString();
        currentServiceData.put(
                LIST_NAME, "iBeacon");
        currentServiceData.put(LIST_UUID, uuid);
        gattServiceData.add(currentServiceData);

        ArrayList<HashMap<String, String>> gattCharacteristicGroupData =
                new ArrayList<HashMap<String, String>>();
        List<BluetoothGattCharacteristic> gattCharacteristics =
                gattService.getCharacteristics();
        ArrayList<BluetoothGattCharacteristic> charas =
                new ArrayList<BluetoothGattCharacteristic>();

        // Loops through available Characteristics.
        for (BluetoothGattCharacteristic gattCharacteristic : gattCharacteristics) {
            charas.add(gattCharacteristic);
            HashMap<String, String> currentCharaData = new HashMap<String, String>();
            uuid = gattCharacteristic.getUuid().toString();
            currentCharaData.put(
                    LIST_NAME, "iBeacon Char");
            currentCharaData.put(LIST_UUID, uuid);
            gattCharacteristicGroupData.add(currentCharaData);
        }
```

```
            mGattCharacteristics.add(charas);
            gattCharacteristicData.add(gattCharacteristicGroupData);
        }

        SimpleExpandableListAdapter gattServiceAdapter = new SimpleExpandableListAdapter(
                this,
                gattServiceData,
                android.R.layout.simple_expandable_list_item_2,
                new String[] {LIST_NAME, LIST_UUID},
                new int[] { android.R.id.text1, android.R.id.text2 },
                gattCharacteristicData,
                android.R.layout.simple_expandable_list_item_2,
                new String[] {LIST_NAME, LIST_UUID},
                new int[] { android.R.id.text1, android.R.id.text2 }
        );
        mGattServicesList.setAdapter(gattServiceAdapter);
    }

    private static IntentFilter makeGattUpdateIntentFilter() {
        final IntentFilter intentFilter = new IntentFilter();
        intentFilter.addAction(BluetoothLeService.ACTION_GATT_CONNECTED);
        intentFilter.addAction(BluetoothLeService.ACTION_GATT_DISCONNECTED);
        intentFilter.addAction(BluetoothLeService.ACTION_GATT_SERVICES_DISCOVERED);
        intentFilter.addAction(BluetoothLeService.ACTION_DATA_AVAILABLE);
        return intentFilter;
    }
}
```

DeviceScanActivity.java

```
package com.example.android.bluetoothlegatt;

import android.app.Activity;
import android.app.ListActivity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothManager;
import android.content.Context;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.os.Bundle;
import android.os.Handler;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

import java.util.ArrayList;

/**
 * Activity for scanning and displaying available Bluetooth LE devices.
 */
public class DeviceScanActivity extends ListActivity {
```

```
private LeDeviceListAdapter mLeDeviceListAdapter;
private BluetoothAdapter mBluetoothAdapter;
private boolean mScanning;
private Handler mHandler;

private static final int REQUEST_ENABLE_BT = 1;
// Stops scanning after 10 seconds.
private static final long SCAN_PERIOD = 10000;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    getActionBar().setTitle(R.string.title_devices);
    mHandler = new Handler();

    // Use this check to determine whether BLE is supported on the device.  Then you can
    // selectively disable BLE-related features.
    if (!getPackageManager().hasSystemFeature(PackageManager.FEATURE_BLUETOOTH_LE)) {
        Toast.makeText(this, R.string.ble_not_supported, Toast.LENGTH_SHORT).show();
        finish();
    }

    // Initializes a Bluetooth adapter.  For API level 18 and above, get a reference to
    // BluetoothAdapter through BluetoothManager.
    final BluetoothManager bluetoothManager =
            (BluetoothManager) getSystemService(Context.BLUETOOTH_SERVICE);
    mBluetoothAdapter = bluetoothManager.getAdapter();

    // Checks if Bluetooth is supported on the device.
    if (mBluetoothAdapter == null) {
        Toast.makeText(this, R.string.error_bluetooth_not_supported, Toast.LENGTH_SHORT).show();
        finish();
        return;
    }
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.main, menu);
    if (!mScanning) {
        menu.findItem(R.id.menu_stop).setVisible(false);
        menu.findItem(R.id.menu_scan).setVisible(true);
        menu.findItem(R.id.menu_refresh).setActionView(null);
    } else {
        menu.findItem(R.id.menu_stop).setVisible(true);
        menu.findItem(R.id.menu_scan).setVisible(false);
        menu.findItem(R.id.menu_refresh).setActionView(
                R.layout.actionbar_indeterminate_progress);
    }
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_scan:
            mLeDeviceListAdapter.clear();
            scanLeDevice(true);
            break;
        case R.id.menu_stop:
```

```
            scanLeDevice(false);
            break;
        }
        return true;
    }

    @Override
    protected void onResume() {
        super.onResume();

        // Ensures Bluetooth is enabled on the device.  If Bluetooth is not currently enabled,
        // fire an intent to display a dialog asking the user to grant permission to enable it.
        if (!mBluetoothAdapter.isEnabled()) {
            if (!mBluetoothAdapter.isEnabled()) {
                Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
                startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
            }
        }

        // Initializes list view adapter.
        mLeDeviceListAdapter = new LeDeviceListAdapter();
        setListAdapter(mLeDeviceListAdapter);
        scanLeDevice(true);
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        // User chose not to enable Bluetooth.
        if (requestCode == REQUEST_ENABLE_BT && resultCode == Activity.RESULT_CANCELED) {
            finish();
            return;
        }
        super.onActivityResult(requestCode, resultCode, data);
    }

    @Override
    protected void onPause() {
        super.onPause();
        scanLeDevice(false);
        mLeDeviceListAdapter.clear();
    }

    @Override
    protected void onListItemClick(ListView l, View v, int position, long id) {
        final BluetoothDevice device = mLeDeviceListAdapter.getDevice(position);
        if (device == null) return;
        //final Intent intent = new Intent(this, DeviceControlActivity.class);
        final Intent intent = new Intent(this, InitOptionSelect.class);
        intent.putExtra(DeviceControlActivity.EXTRAS_DEVICE_NAME, device.getName());
        intent.putExtra(DeviceControlActivity.EXTRAS_DEVICE_ADDRESS, device.getAddress());
        if (mScanning) {
            mBluetoothAdapter.stopLeScan(mLeScanCallback);
            mScanning = false;
        }
        startActivity(intent);
    }

    private void scanLeDevice(final boolean enable) {
        if (enable) {
            // Stops scanning after a pre-defined scan period.
```

```
            mHandler.postDelayed(new Runnable() {
                @Override
                public void run() {
                    mScanning = false;
                    mBluetoothAdapter.stopLeScan(mLeScanCallback);
                    invalidateOptionsMenu();
                }
            }, SCAN_PERIOD);

            mScanning = true;
            mBluetoothAdapter.startLeScan(mLeScanCallback);
        } else {
            mScanning = false;
            mBluetoothAdapter.stopLeScan(mLeScanCallback);
        }
        invalidateOptionsMenu();
    }

    // Adapter for holding devices found through scanning.
    private class LeDeviceListAdapter extends BaseAdapter {
        private ArrayList<BluetoothDevice> mLeDevices;
        private LayoutInflater mInflator;

        public LeDeviceListAdapter() {
            super();
            mLeDevices = new ArrayList<BluetoothDevice>();
            mInflator = DeviceScanActivity.this.getLayoutInflater();
        }

        public void addDevice(BluetoothDevice device) {
            if(!mLeDevices.contains(device)) {
                mLeDevices.add(device);
            }
        }

        public BluetoothDevice getDevice(int position) {
            return mLeDevices.get(position);
        }

        public void clear() {
            mLeDevices.clear();
        }

        @Override
        public int getCount() {
            return mLeDevices.size();
        }

        @Override
        public Object getItem(int i) {
            return mLeDevices.get(i);
        }

        @Override
        public long getItemId(int i) {
            return i;
        }

        @Override
        public View getView(int i, View view, ViewGroup viewGroup) {
```

```
            ViewHolder viewHolder;
            // General ListView optimization code.
            if (view == null) {
                view = mInflator.inflate(R.layout.listitem_device, null);
                viewHolder = new ViewHolder();
                viewHolder.deviceAddress = (TextView) view.findViewById(R.id.device_address);
                viewHolder.deviceName = (TextView) view.findViewById(R.id.device_name);
                view.setTag(viewHolder);
            } else {
                viewHolder = (ViewHolder) view.getTag();
            }

            BluetoothDevice device = mLeDevices.get(i);
            final String deviceName = device.getName();
            if (deviceName != null && deviceName.length() > 0)
                viewHolder.deviceName.setText(deviceName);
            else
                viewHolder.deviceName.setText(R.string.unknown_device);
            viewHolder.deviceAddress.setText(device.getAddress());

            return view;
        }
    }

    // Device scan callback.
    private BluetoothAdapter.LeScanCallback mLeScanCallback =
            new BluetoothAdapter.LeScanCallback() {

        @Override
        public void onLeScan(final BluetoothDevice device, int rssi, byte[] scanRecord) {
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    mLeDeviceListAdapter.addDevice(device);
                    mLeDeviceListAdapter.notifyDataSetChanged();
                }
            });
        }
    };

    static class ViewHolder {
        TextView deviceName;
        TextView deviceAddress;
    }
}
```

GraphViewData.java

```
package com.example.android.bluetoothlegatt;

import com.jjoe64.graphview.GraphViewDataInterface;

public class GraphViewData implements GraphViewDataInterface {
            private double x, y;

            public GraphViewData(double x, double y) {
                        this.x = x;
                        this.y = y;
```

```
	}

	@Override
	public double getX() {
		return this.x;
	}

	@Override
	public double getY() {
		return this.y;
	}

//	public int compareTo(GraphViewData other) {
//
//		if(this.getX() < other.getX()){
//			return -1;
//		}
//		if(this.getX() == other.getX()){
//			return 0;
//		}
//
//		return 1;
//	}
}
```

InitOptionSelect.java

```
package com.example.android.bluetoothlegatt;

import java.io.FileWriter;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.HashMap;
import java.util.Hashtable;
import java.util.List;
import java.util.TimeZone;
import java.util.UUID;

import com.opencsv.CSVWriter;

import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothGatt;
import android.bluetooth.BluetoothGattCallback;
import android.bluetooth.BluetoothGattCharacteristic;
import android.bluetooth.BluetoothGattDescriptor;
import android.bluetooth.BluetoothGattService;
import android.bluetooth.BluetoothManager;
import android.content.BroadcastReceiver;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.ServiceConnection;
```

```
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.ExpandableListView;
import android.widget.SimpleExpandableListAdapter;
import android.widget.TextView;

public class InitOptionSelect extends Activity {

	private static CSVWriter writer = null;

	private static final UUID FRAM_BUFFER = UUID
			.fromString("f000aa41-0451-4000-b000-000000000000");
	private static final UUID ADXL_DATA = UUID
			.fromString("f000aa11-0451-4000-b000-000000000000");
	private static final UUID LIGHT_DATA = UUID
			.fromString("f000aa21-0451-4000-b000-000000000000");
	private static final UUID TEMP_DATA = UUID
			.fromString("f000aa01-0451-4000-b000-000000000000");

	private static final UUID FRAM_CONFIG = UUID
			.fromString("f000aa42-0451-4000-b000-000000000000");

	private static final UUID TIME_SERVICE = UUID
			.fromString("f000dd10-0451-4000-b000-000000000000");
	private static final UUID TIME_MONTH_CONFIG = UUID
			.fromString("f000dd13-0451-4000-b000-000000000000");
	private static final UUID TIME_DAY_CONFIG = UUID
			.fromString("f000dd14-0451-4000-b000-000000000000");
	private static final UUID TIME_YEAR_CONFIG = UUID
			.fromString("f000dd15-0451-4000-b000-000000000000");

	private final static String TAG = DeviceControlActivity.class
			.getSimpleName();

	public static final String EXTRAS_DEVICE_NAME = "DEVICE_NAME";
	public static final String EXTRAS_DEVICE_ADDRESS = "DEVICE_ADDRESS";

	private BluetoothAdapter mBtAdapter;
	private BluetoothGatt gatt;
	private BluetoothDevice device;
	private TextView mConnectionState;
	private TextView data_sync_text;
	private TextView mDataField;
	private TextView _realTimeXdata;
	private TextView _realTimeYdata;
	private TextView _realTimeZdata;
	private TextView _realTimeVISdata;
	private TextView _realTimeIRdata;
	private TextView _realTimeUVdata;
	private TextView _realTimeTempdata;
	private String mDeviceName;
	private String mDeviceAddress;
	private ExpandableListView mGattServicesList;
	private BluetoothLeService mBluetoothLeService;
	private ArrayList<BluetoothGattCharacteristic> mGattCharacteristics = new
ArrayList<BluetoothGattCharacteristic>();
```

```
	private BluetoothGattCharacteristic characteristic;

	private Hashtable<String, BluetoothGattCharacteristic> characteristicsByUUID = new Hashtable<String,
BluetoothGattCharacteristic>();

	private boolean mConnected = false;
	private BluetoothGattCharacteristic mNotifyCharacteristic;

	private final String LIST_NAME = "NAME";
	private final String LIST_UUID = "UUID";

	private Handler mHandler;
	private int sync_byte = 0;
	private int timerSends = 0;
	private int month;
	private int day;
	private int year;
	private int hour;
	private int min;

	private String data_string;
	private static View thisView;

	private ArrayList<ADXL_Obj> ADXL_DATA_HOLDER = new ArrayList<ADXL_Obj>();
	private ArrayList<LIGHT_Obj> LIGHT_DATA_HOLDER = new ArrayList<LIGHT_Obj>();
	private ArrayList<TMP006_Obj> TMP006_DATA_HOLDER = new ArrayList<TMP006_Obj>();
	private ArrayList<BPM_Obj> BPM_DATA_HOLDER = new ArrayList<BPM_Obj>();

	private Intent intent;

	private boolean realTime = false;

	public static final int FORMAT_UINT8 = 17;
	public static final int FORMAT_SINT8 = 33;

	private final ServiceConnection mServiceConnection = new ServiceConnection() {

		@Override
		public void onServiceConnected(ComponentName componentName,
				IBinder service) {
			mBluetoothLeService = ((BluetoothLeService.LocalBinder) service)
					.getService();
			mBluetoothLeService.initialize();
			if (!mBluetoothLeService.initialize()) {
				Log.e(TAG, "Unable to initialize Bluetooth");
				finish();
			}
			// Automatically connects to the device upon successful start-up
			// initialization.
			mBluetoothLeService.connect(mDeviceAddress);

		}

		@Override
		public void onServiceDisconnected(ComponentName componentName) {
			mBluetoothLeService = null;
		}
	};

	// Handles various events fired by the Service.
```

```
// ACTION_GATT_CONNECTED: connected to a GATT server.
// ACTION_GATT_DISCONNECTED: disconnected from a GATT server.
// ACTION_GATT_SERVICES_DISCOVERED: discovered GATT services.
// ACTION_DATA_AVAILABLE: received data from the device. This can be a
// result of read
// or notification operations.
private final BroadcastReceiver mGattUpdateReceiver = new BroadcastReceiver() {
	@Override
	public void onReceive(Context context, Intent intent) {
		final String action = intent.getAction();
		if (BluetoothLeService.ACTION_GATT_CONNECTED.equals(action)) {
			Log.i("ON GATT CONNECTED 1", "OK");
			mConnected = true;
			updateConnectionState(R.string.connected);
			invalidateOptionsMenu();
			Log.i("ON GATT CONNECTED 2", "OK");
		} else if (BluetoothLeService.ACTION_GATT_DISCONNECTED
				.equals(action)) {
			mConnected = false;
			updateConnectionState(R.string.disconnected);
			invalidateOptionsMenu();
			clearUI();
			Log.i("ON GATT DISCONNECTED", "OK");
		} else if (BluetoothLeService.ACTION_GATT_SERVICES_DISCOVERED
				.equals(action)) {
			// Show all the supported services and characteristics on the
			// user interface.
			Log.i("ON GATT SERVICE DISCOVERED", "OK");
			getGattServices(mBluetoothLeService.getSupportedGattServices());
			Log.i("ON DISPLAYED SERVICES", "OK");
		} else if (BluetoothLeService.ACTION_DATA_AVAILABLE.equals(action)) {
			Log.i("ON GATT DATA AVAILABLE", "OK");
			displayData(intent
					.getStringExtra(BluetoothLeService.EXTRA_DATA));

		}
	}
};

private void updateConnectionState(final int resourceId) {
	runOnUiThread(new Runnable() {
		@Override
		public void run() {
			mConnectionState.setText(resourceId);
		}
	});
}

private void displayData(String data) {
	if (data != null) {

		if (data.length() == 45) {
			// Log.d(TAG, "" + data.length());
			if (realTime == false) {
				mDataField.setText(data.substring(data.length() - 34,
						data.length() - 1).replace(" ", ":"));

			} else {
				mDataField.setText("REAL TIME MODE");
			}
```

```
                        sortData(data.substring(data.length() - 34, data.length() - 1)
                                .replace(" ", ":"));

                } else {

                        mDataField.setText(data);
                }

        }
}

private void sortData(String data) {

        Log.d(TAG, data);
        String[] delimitString = data.split(":");

        Log.d(TAG, delimitString[0]);

        Integer header = Integer.parseInt("" + delimitString[0].charAt(2));

        String UTCtime = delimitString[1] + "" + delimitString[2] + ""
                        + delimitString[3] + "" + delimitString[4];

        Integer UTC = Integer.parseInt(UTCtime, 16);

        Integer corrected_UTC = (UTC - 86400);

        // long ts = (long) UTC;
        // Date localTime = new Date(ts);
        // String format = "yyyy/MM/dd HH:mm:ss";
        // SimpleDateFormat sdf = new SimpleDateFormat(format);
        // Log.d(TAG, "" + UTC);

        // String dateAsText = new
        // SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(new
        // Date(Integer.parseInt(UTCtime, 16) * 1000L));
        Log.d(TAG, "" + getDateCurrentTimeZone((long) UTC));

        data_string = delimitString[5] + ":" + delimitString[6] + ":"
                        + delimitString[7] + ":" + delimitString[8] + ":"
                        + delimitString[9] + ":" + delimitString[10];
        if (UTC != 0) {
                switch (header) {

                case 0: // empty
                        Log.d(TAG, getDateCurrentTimeZone((long) UTC) + " EMPTY DATA: "
                                        + data_string);
                        break;

                case 1: // ADXL
                        Log.d(TAG, "ADXL DATA: " + data_string);
                        ADXL_Obj adxl_data = new ADXL_Obj(UTC, data_string);
                        if (realTime == true) {
                                _realTimeXdata.setText("X Data: " + adxl_data.getX()
                                                + " m/s^2");
                                _realTimeYdata.setText("Y Data: " + adxl_data.getY()
                                                + " m/s^2");
                                _realTimeZdata.setText("Z Data: " + adxl_data.getZ()
                                                + " m/s^2");
```

```
                } else {
                        // ADXL_DATA_HOLDER.add(adxl_data);
                        double mag = adxl_data.getMagnitude();
                        if (mag == 0) {
                                break;
                        }
                        // CSV CSV CSV
                        String first = "Accelerometer";
                        String second = getDateCurrentTimeZone(corrected_UTC);
                        String third = String.valueOf(mag);
                        String fourth = String.valueOf(UTC);
                        String[] entries = (first + "," + second + "," + third
                                        + "," + fourth + ",").split(",");
                        writer.writeNext(entries);
                }
                break;
        case 2: // LIGHT

                Log.d(TAG, getDateCurrentTimeZone((long) UTC) + " LIGHT DATA: "
                                + data_string);
                LIGHT_Obj light_data = new LIGHT_Obj(UTC, data_string);
                if (realTime == true) {
                        _realTimeVISdata.setText("Visible Light: "
                                        + light_data.getVIS() + " lux");
                        _realTimeIRdata.setText("IR Light: " + light_data.getIR()
                                        + " lux");
                        _realTimeUVdata.setText("UV Light: " + light_data.getUV());
                } else {
                        float visible = light_data.getVIS();
                        float ir = light_data.getIR();
                        float uv = light_data.getUV();
                        // Check for small values
                        if ((visible <= 0.001) || (ir <= 0.001) || (uv <= 0.001)) {
                                break;
                        }
                        // LIGHT_DATA_HOLDER.add(light_data);
                        // CSV CSV CSV
                        String first = "Visible";
                        String second = getDateCurrentTimeZone(corrected_UTC);
                        String third = Float.toString(visible);
                        String fourth = String.valueOf(UTC);
                        String[] entries = (first + "," + second + "," + third
                                        + "," + fourth + ",").split(",");
                        writer.writeNext(entries);
                        // CSV CSV CSV
                        String first2 = "IR";
                        String second2 = getDateCurrentTimeZone(corrected_UTC);
                        String third2 = Float.toString(ir);
                        String fourth2 = String.valueOf(UTC);
                        String[] entries2 = (first2 + "," + second2 + "," + third2
                                        + "," + fourth2).split(",");
                        writer.writeNext(entries2);
                        // CSV CSV CSV
                        String first3 = "UV";
                        String second3 = getDateCurrentTimeZone(corrected_UTC);
                        String third3 = Float.toString(uv);
                        String fourth3 = String.valueOf(UTC);
                        String[] entries3 = (first3 + "," + second3 + "," + third3
                                        + "," + fourth3).split(",");
                        writer.writeNext(entries3);
```

```
                        }
                        break;
                case 3: // BPM
                        Log.d(TAG, getDateCurrentTimeZone((long) UTC) + " BPM DATA: "
                                        + data_string);

                        String BPM_String = data_string.replace(":", "");
                        Integer BPM_Integer = Integer.parseInt(BPM_String, 16);
                        BPM_Obj bpm_data = new BPM_Obj(UTC, data_string);
                        // BPM_DATA_HOLDER.add(bpm_data);
                        // CSV CSV CSV
                        String first = "BPM";
                        String second = getDateCurrentTimeZone(corrected_UTC);
                        String third = String.valueOf(BPM_Integer);
                        String fourth = String.valueOf(UTC);
                        String[] entries = (first + "," + second + "," + third + ","
                                        + fourth + ",").split(",");
                        writer.writeNext(entries);
                        break;

                case 4: // TMP006
                        Log.d(TAG, getDateCurrentTimeZone((long) UTC)
                                        + " TMP006 DATA: " + data_string);
                        TMP006_Obj tmp_data = new TMP006_Obj(UTC, data_string);
                        Log.d(TAG, "TMP006 DATA ACTUAL TEMP: " + tmp_data.getTemp());

                        if (realTime == true) {
                                if ((Double.isNaN(tmp_data.getTemp()))
                                                || (tmp_data.getTemp() == 0)) {
                                        break;
                                }
                                _realTimeTempdata.setText("Temperature: "
                                                + tmp_data.getTemp() + " F");
                        } else {
                                // TMP006_DATA_HOLDER.add(tmp_data);
                                double temp = tmp_data.getTemp();
                                if ((Double.isNaN(temp)) || (temp == 0)) {
                                        break;
                                }
                                // CSV CSV CSV
                                String first1 = "Temperature";
                                String second1 = getDateCurrentTimeZone(corrected_UTC);
                                String third1 = String.valueOf(temp);
                                String fourth1 = String.valueOf(UTC);
                                String[] entries1 = (first1 + "," + second1 + "," + third1
                                                + "," + fourth1 + ",").split(",");
                                writer.writeNext(entries1);
                        }
                        break;

                }
        }

}

private String getDateCurrentTimeZone(long timestamp) {
        try {
                Calendar calendar = Calendar.getInstance();
                TimeZone tz = TimeZone.getDefault();
                calendar.setTimeInMillis(timestamp * 1000);
```

```
                        calendar.add(Calendar.MILLISECOND, 0);
                        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
                        Date currenTimeZone = (Date) calendar.getTime();
                        return sdf.format(currenTimeZone);
                } catch (Exception e) {
                }
                return "";
        }

        private void clearUI() {
                mGattServicesList.setAdapter((SimpleExpandableListAdapter) null);
                mDataField.setText(R.string.no_data);
        }

        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.init_option_select);

                intent = getIntent();

                mDeviceName = intent.getStringExtra(EXTRAS_DEVICE_NAME);
                mDeviceAddress = intent.getStringExtra(EXTRAS_DEVICE_ADDRESS);

                mConnectionState = (TextView) findViewById(R.id.connection_state);

                // Sets up UI references.
                ((TextView) findViewById(R.id.device_address)).setText(mDeviceAddress);

                Intent gattServiceIntent = new Intent(this, BluetoothLeService.class);
                bindService(gattServiceIntent, mServiceConnection, BIND_AUTO_CREATE);

                final BluetoothManager bluetoothManager = (BluetoothManager)
getSystemService(Context.BLUETOOTH_SERVICE);
                mBtAdapter = bluetoothManager.getAdapter();

                device = mBtAdapter.getRemoteDevice(mDeviceAddress);

                gatt = device.connectGatt(this, false, mGattCallback);

                _realTimeXdata = ((TextView) findViewById(R.id.xData));
                _realTimeYdata = ((TextView) findViewById(R.id.yData));
                _realTimeZdata = ((TextView) findViewById(R.id.zData));
                _realTimeVISdata = ((TextView) findViewById(R.id.visibleData));
                _realTimeIRdata = ((TextView) findViewById(R.id.irData));
                _realTimeUVdata = ((TextView) findViewById(R.id.uvData));
                _realTimeTempdata = ((TextView) findViewById(R.id.temperature));

                final Button button_syncTime = (Button) findViewById(R.id.SyncTime);
                final Button button_syncData = (Button) findViewById(R.id.SyncData);
                final Button button_RealTime = (Button) findViewById(R.id.RealTime);

                // intent = new Intent(this, Grapher.class);

                button_syncTime.setOnClickListener(new View.OnClickListener() {
                        public void onClick(View v) {

                                button_syncData.setEnabled(false);

                                Calendar c = Calendar.getInstance();
                                month = c.get(Calendar.MONTH);
```

```
                        Log.d(TAG, "MONTH: " + (byte) month);
                        day = c.get(Calendar.DAY_OF_MONTH);
                        String year_string = "" + c.get(Calendar.YEAR);
                        year_string = year_string.substring(year_string.length() - 2,
                                        year_string.length());
                        Log.d(TAG, year_string);
                        year = Integer.parseInt(year_string);
                        hour = c.get(Calendar.HOUR_OF_DAY);
                        min = c.get(Calendar.MINUTE);

                        Log.d(TAG, "SYNC TIME PRESSED");
                        mDataField = (TextView) findViewById(R.id.data_value);

                        mDataField.setText("" + (byte) month + " " + (byte) day + " "
                                        + (byte) year + " " + (byte) hour + " " + (byte) min);

                        sync_timer();

                }
        });

        button_syncData.setOnClickListener(new View.OnClickListener() {
                public void onClick(View v) {

                        button_syncTime.setEnabled(false);
                        button_syncData.setEnabled(false);
                        button_RealTime.setEnabled(false);

                        Date date = new Date();
                        String csv_name = new SimpleDateFormat("yyyy.MM.dd  HH:mm")
                                        .format(date);
                        try {
                                writer = new CSVWriter(new FileWriter("/sdcard/" + csv_name
                                                + "-circadianrhythm.csv"), ',');
                        } catch (IOException e) {
                                // error
                        }

                        data_sync_text = (TextView) InitOptionSelect.this
                                        .findViewById(R.id.data_sync_text);
                        Log.d(TAG, "SYNC Data PRESSED");
                        mDataField = (TextView) findViewById(R.id.data_value);

                        readCharacteristicValue(FRAM_BUFFER);

                        data_sync_text.setVisibility(v.VISIBLE);

                        InitOptionSelect.this.findViewById(R.id.data_sync_progressbar)
                                        .setVisibility(v.VISIBLE);

                        thisView = v;

                        realTime = false;

                        sync_counter();

                }
        });

        button_RealTime.setOnClickListener(new View.OnClickListener() {
```

```
                public void onClick(View v) {

                        button_syncTime.setEnabled(false);
                        button_syncData.setEnabled(false);

                        realTime = true;

                        Log.d(TAG, "FRAM PRESSED");
                        mDataField = (TextView) findViewById(R.id.data_value);

                        _realTimeXdata.setVisibility(v.VISIBLE);
                        _realTimeYdata.setVisibility(v.VISIBLE);
                        _realTimeZdata.setVisibility(v.VISIBLE);
                        _realTimeVISdata.setVisibility(v.VISIBLE);
                        _realTimeIRdata.setVisibility(v.VISIBLE);
                        _realTimeUVdata.setVisibility(v.VISIBLE);
                        // characteristic = getCharac(FRAM_CONFIG);
                        // characteristic.setValue(new byte[] {0x01});
                        // gatt.writeCharacteristic(characteristic);

                        real_time();

                        // Log.d(TAG,"CHARACTERISTIC IS " + characteristic.getUuid());

                }
        });

        /*
         * mGattServicesList = (ExpandableListView)
         * findViewById(R.id.gatt_services_list);
         * //mGattServicesList.setOnChildClickListener
         * (servicesListClickListner);
         *
         * mDataField = (TextView) findViewById(R.id.data_value);
         *
         * getActionBar().setTitle(mDeviceName);
         * getActionBar().setDisplayHomeAsUpEnabled(true); Intent
         * gattServiceIntent = new Intent(this, BluetoothLeService.class);
         * bindService(gattServiceIntent, mServiceConnection, BIND_AUTO_CREATE);
         */
}

private BluetoothGattCallback mGattCallback = new BluetoothGattCallback() {

};

@Override
protected void onResume() {
        super.onResume();
        registerReceiver(mGattUpdateReceiver, makeGattUpdateIntentFilter());
        if (mBluetoothLeService != null) {
                final boolean result = mBluetoothLeService.connect(mDeviceAddress);
                Log.d(TAG, "Connect request result=" + result);
        }
}

@Override
protected void onPause() {
        super.onPause();
        unregisterReceiver(mGattUpdateReceiver);
```

```
	}

	@Override
	protected void onDestroy() {
		super.onDestroy();
		unbindService(mServiceConnection);
		mBluetoothLeService = null;
	}

	private static IntentFilter makeGattUpdateIntentFilter() {
		final IntentFilter intentFilter = new IntentFilter();
		intentFilter.addAction(BluetoothLeService.ACTION_GATT_CONNECTED);
		intentFilter.addAction(BluetoothLeService.ACTION_GATT_DISCONNECTED);
		intentFilter
				.addAction(BluetoothLeService.ACTION_GATT_SERVICES_DISCOVERED);
		intentFilter.addAction(BluetoothLeService.ACTION_DATA_AVAILABLE);
		return intentFilter;
	}

	private void getGattServices(List<BluetoothGattService> gattServices) {
		if (gattServices == null)
			return;
		String uuid = null;
		ArrayList<HashMap<String, String>> gattServiceData = new ArrayList<HashMap<String,
String>>();
		ArrayList<ArrayList<HashMap<String, String>>> gattCharacteristicData = new
ArrayList<ArrayList<HashMap<String, String>>>();
		mGattCharacteristics = new ArrayList<BluetoothGattCharacteristic>();

		// Loops through available GATT Services.
		for (BluetoothGattService gattService : gattServices) {
			HashMap<String, String> currentServiceData = new HashMap<String, String>();
			uuid = gattService.getUuid().toString();
			currentServiceData.put(LIST_NAME, "iBeacon");
			currentServiceData.put(LIST_UUID, uuid);
			gattServiceData.add(currentServiceData);

			ArrayList<HashMap<String, String>> gattCharacteristicGroupData = new
ArrayList<HashMap<String, String>>();
			List<BluetoothGattCharacteristic> gattCharacteristics = gattService
					.getCharacteristics();
			ArrayList<BluetoothGattCharacteristic> charas = new
ArrayList<BluetoothGattCharacteristic>();

			// Loops through available Characteristics.
			for (BluetoothGattCharacteristic gattCharacteristic : gattCharacteristics) {
				characteristicsByUUID.put(gattCharacteristic.getUuid()
						.toString(), gattCharacteristic);
				Log.d(TAG, "" + gattCharacteristic.getUuid() + " "
						+ gattCharacteristic.getPermissions());
				charas.add(gattCharacteristic);
				HashMap<String, String> currentCharaData = new HashMap<String,
String>();
				uuid = gattCharacteristic.getUuid().toString();
				// Log.d(TAG,"" + uuid);
				currentCharaData.put(LIST_NAME, "iBeacon Char");
				currentCharaData.put(LIST_UUID, uuid);
				gattCharacteristicGroupData.add(currentCharaData);
			}
			// mGattCharacteristics.add(charas);
```

```
                gattCharacteristicData.add(gattCharacteristicGroupData);
        }

}

private BluetoothGattCharacteristic getCharac(UUID id) {
        for (BluetoothGattService gattService : mBluetoothLeService
                        .getSupportedGattServices()) {

                List<BluetoothGattCharacteristic> gattCharacteristics = gattService
                                .getCharacteristics();

                for (BluetoothGattCharacteristic b : gattCharacteristics) {

                        if (b.getUuid().equals(id)) {
                                Log.d(TAG, "" + b.getUuid());
                                characteristic = b;
                                return characteristic;
                        }
                }

        }
        return characteristic;

}

private void readCharacteristicValue(UUID id) {
        for (BluetoothGattService gattService : mBluetoothLeService
                        .getSupportedGattServices()) {

                List<BluetoothGattCharacteristic> gattCharacteristics = gattService
                                .getCharacteristics();

                for (BluetoothGattCharacteristic b : gattCharacteristics) {

                        if (b.getUuid().equals(id)) {
                                Log.d(TAG, "" + b.getUuid());
                                characteristic = b;
                        }
                }

        }

        if (characteristic != null) {

                Log.d(TAG, "NOT NULL");

                mBluetoothLeService.readCharacteristic(characteristic);

                int charaProp = characteristic.getProperties();
                if ((charaProp | BluetoothGattCharacteristic.PROPERTY_READ) > 0) {
                        // If there is an active notification on a characteristic, clear
                        // it first so it doesn't update the data field on the user
                        // interface.
                        if (mNotifyCharacteristic != null) {
                                mBluetoothLeService.setCharacteristicNotification(
                                                mNotifyCharacteristic, false);
                                mNotifyCharacteristic = null;
                        }
                        mBluetoothLeService.readCharacteristic(characteristic);
```

```
                        }
                        if ((charaProp | BluetoothGattCharacteristic.PROPERTY_NOTIFY) > 0) {
                                mNotifyCharacteristic = characteristic;
                                mBluetoothLeService.setCharacteristicNotification(
                                                characteristic, true);

                        }

                        // mDataField.setText("" + characteristic.getUuid());

                }
        }

        private void real_time() {
                final Handler handler = new Handler();
                Runnable runnable = new Runnable() {
                        public void run() {
                                while (realTime == true) {
                                        try {
                                                Thread.sleep(1000);
                                        } catch (InterruptedException e) {
                                                e.printStackTrace();
                                        }
                                        handler.post(new Runnable() {
                                                public void run() {
                                                        readCharacteristicValue(ADXL_DATA);
                                                        try {
                                                                Thread.sleep(100);
                                                        } catch (InterruptedException e) {
                                                                // TODO Auto-generated catch block
                                                                e.printStackTrace();
                                                        }
                                                        readCharacteristicValue(LIGHT_DATA);
                                                        try {
                                                                Thread.sleep(100);
                                                        } catch (InterruptedException e) {
                                                                // TODO Auto-generated catch block
                                                                e.printStackTrace();
                                                        }
                                                        readCharacteristicValue(TEMP_DATA);

                                                }
                                        });
                                }
                        }
                };
                new Thread(runnable).start();
        }

        private void sync_counter() {
                final Handler handler = new Handler();
                Runnable runnable = new Runnable() {
                        public void run() {
                                while (true) {
                                        try {
                                                Thread.sleep(500);
                                        } catch (InterruptedException e) {
                                                e.printStackTrace();
                                        }
                                        handler.post(new Runnable() {
```

```
                                                    public void run() {
                                                            data_sync_text.setText("Synching Data... Block "
                                                                    + sync_byte + " of 5957");
                                                            if (sync_byte < 50) {
                                                                    characteristic =
getCharac(FRAM_CONFIG);
                                                                    if (characteristic != null) {

                                                                            characteristic
                                                                                    .setValue(new
byte[] { 0x01 });

gatt.writeCharacteristic(characteristic);

                                                                            try {
                                                                                    Thread.sleep(100);
                                                                            } catch (InterruptedException e) {
                                                                                    e.printStackTrace();
                                                                            }


readCharacteristicValue(FRAM_BUFFER);

                                                                            sync_byte++;

                                                                            if (sync_byte == 50) {

                                                                                    try {
                                                                                            writer.close();
                                                                                    } catch (IOException e)
{
                                                                                            // TODO
Auto-generated catch block

e.printStackTrace();
                                                                                    }

                                                                                    data_sync_text

.setVisibility(thisView.INVISIBLE);


InitOptionSelect.this.findViewById(

R.id.data_sync_progressbar)

.setVisibility(

                        thisView.INVISIBLE);

intent.putExtra("adxl_data",

ADXL_DATA_HOLDER);

intent.putExtra("light_data",

LIGHT_DATA_HOLDER);

intent.putExtra("bpm_data",
```

```
BPM_DATA_HOLDER);

intent.putExtra("tmp006_data",

TMP006_DATA_HOLDER);
                                                                                    // startActivity(intent);

                                                                                }
                                                                            }
                                                                        }
                                                                    }
                                                                });
                                                            }
                                                        }
                                                    };
                                                    new Thread(runnable).start();
                                                }

                                                private void sync_timer() {

                                                    final Handler handler = new Handler();
                                                    Runnable runnable = new Runnable() {
                                                        public void run() {
                                                            while (timerSends <= 5) {
                                                                try {
                                                                    Thread.sleep(500);
                                                                } catch (InterruptedException e) {
                                                                    e.printStackTrace();
                                                                }
                                                                handler.post(new Runnable() {
                                                                    public void run() {

                                                                        characteristic =
getCharac(TIME_MONTH_CONFIG);
                                                                        if (characteristic != null) {
                                                                            Log.d(TAG, "CHARACTERISTIC IS "
                                                                                    +
characteristic.getUuid());
                                                                            switch (timerSends) {

                                                                            case 0:
                                                                                BluetoothGattDescriptor desc =
characteristic

.getDescriptor(TIME_DAY_CONFIG);
                                                                                desc.setValue(new byte[] { (byte)
month });
                                                                                gatt.writeDescriptor(desc);
                                                                                break;

                                                                            case 1:
                                                                                desc = characteristic

.getDescriptor(TIME_DAY_CONFIG);
                                                                                desc.setValue(new byte[] { (byte)
day });
                                                                                gatt.writeDescriptor(desc);
                                                                                break;
```

```
                                                                    case 2:
                                                                            desc = characteristic
.getDescriptor(TIME_DAY_CONFIG);
                                                                            desc.setValue(new byte[] { (byte)
year });
                                                                            gatt.writeDescriptor(desc);
                                                                            break;

                                                                    case 3:
                                                                            desc = characteristic
.getDescriptor(TIME_DAY_CONFIG);
                                                                            desc.setValue(new byte[] { (byte)
hour });
                                                                            gatt.writeDescriptor(desc);
                                                                            break;

                                                                    case 4:
                                                                            desc = characteristic
.getDescriptor(TIME_DAY_CONFIG);
                                                                            desc.setValue(new byte[] { (byte)
min });
                                                                            gatt.writeDescriptor(desc);
                                                                            break;

                                                                    }

                                                                    timerSends++;
                                                            }
                                                    }
                                            });
                                    }
                            }
                    };
                    new Thread(runnable).start();
            }
            private void createCSV() {
            }
}
```

IRLIGHT_GRAPHER.java

```
package com.example.android.bluetoothlegatt;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.TimeZone;
```

```
import com.jjoe64.graphview.CustomLabelFormatter;
import com.jjoe64.graphview.GraphView;
import com.jjoe64.graphview.GraphView.GraphViewData;
import com.jjoe64.graphview.GraphViewSeries;
import com.jjoe64.graphview.LineGraphView;

import android.app.Activity;
import android.os.Bundle;
import android.os.Environment;
import android.util.Log;
import android.widget.LinearLayout;

public class IRLIGHT_GRAPHER extends Activity {

	private final static String TAG = IRLIGHT_GRAPHER.class.getSimpleName();

	// String file = (String)
	// getIntent().getSerializableExtra("EXTRAS_FILE_NAME");
	// String file = "/sdcard/abc.csv";
	public static String file;

	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.test);

		Bundle extras = IRLIGHT_GRAPHER.this.getIntent().getExtras();

		ArrayList<GraphViewData> data = new ArrayList<GraphViewData>();

		file = extras.getString("EXTRAS_FILE_NAME");

		file = Environment.getExternalStorageDirectory() + "/" + file;
		// file = "/sdcard/" + file;

		BufferedReader reader = null;
		try {
			reader = new BufferedReader(new FileReader(file));
		} catch (FileNotFoundException e1) {
			// TODO Auto-generated catch block
			e1.printStackTrace();
		}
		try {
			String line;
			while ((line = reader.readLine()) != null) {
				String[] RowData = line.split(",");
				String type = RowData[0].replace("", ' ').trim();

				String date = RowData[1].replace("", ' ').trim();

				String light = RowData[2].replace("", ' ').trim();

				String utc = RowData[3].replace("", ' ').trim();

				if (type.equals("IR")) {

					Log.d(TAG, "IN HERE " + type);
					Log.d(TAG, "IN HERE UTC" + (double) Long.parseLong(utc));
					Log.d(TAG, "IN HERE IRLIGHT " + Double.parseDouble(light));

					data.add(new GraphViewData((double) Long.parseLong(utc),
```

```
                                                    Double.parseDouble(light)));

                                }
                        }
                } catch (IOException ex) {
                        // handle exception
                } finally {
                        try {
                                reader.close();
                        } catch (IOException e) {
                                // handle exception
                        }
                }

                GraphView graphView = new LineGraphView(this, "IR Light");

                graphView.setCustomLabelFormatter(new CustomLabelFormatter() {

                        @Override
                        public String formatLabel(double value, boolean isValueX) {
                                if (isValueX) {
                                        return getDateCurrentTimeZone((long) value);
                                }
                                return null;
                        }
                });

                GraphViewData[] input = new GraphViewData[data.size()];

                for (int i = 0; i < input.length; i++) {

                        input[i] = data.get(i);
                }

                // add data
                graphView.addSeries(new GraphViewSeries(input));
                // set view port, start=2, size=40
                graphView.setViewPort(2, 40);
                graphView.setScrollable(true);
                // optional - activate scaling / zooming
                graphView.setScalable(true);

                LinearLayout layout = (LinearLayout) findViewById(R.id.graph1);
                layout.addView(graphView);
        }

        private String getDateCurrentTimeZone(long timestamp) {
                try {
                        Calendar calendar = Calendar.getInstance();
                        TimeZone tz = TimeZone.getDefault();
                        calendar.setTimeInMillis(timestamp * 1000);
                        calendar.add(Calendar.MILLISECOND, 0);
                        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
                        Date currenTimeZone = (Date) calendar.getTime();
                        return sdf.format(currenTimeZone);
                } catch (Exception e) {
                }
                return "";
        }
```

```
            /*
             * int num = 150;
             *
             * double v=0; for (int i=0; i<num; i++) { v += 0.2;
             *
             * }
             */
}
```

LIGHT_Obj.java

```
package com.example.android.bluetoothlegatt;

import java.io.Serializable;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.TimeZone;

public class LIGHT_Obj implements Serializable{

            private String date;
            private String data;

            private float UV;
            private float IR;
            private float VIS;

            public LIGHT_Obj(Integer UTC, String data){

                        this.date = getDateCurrentTimeZone(UTC);
                        this.data = data;

            }

            private String getDateCurrentTimeZone(long timestamp) {
      try{
         Calendar calendar = Calendar.getInstance();
         TimeZone tz = TimeZone.getDefault();
         calendar.setTimeInMillis(timestamp * 1000);
         calendar.add(Calendar.MILLISECOND, 0);
         SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
         Date currenTimeZone = (Date) calendar.getTime();
         return sdf.format(currenTimeZone);
      }catch (Exception e) {
      }
      return "";
   }

            public String getDate() {
                        return date;
            }

            public String getData() {
                        return data;
            }

            public float getUV() {
```

```
                String[] delimit = data.split(":");
                String IRString = delimit[4] + "" + delimit[5];
                Integer decVal = Integer.parseInt(IRString, 16)
                return (float)decVal/(float)100;
        }
        public float getIR() {
                String[] delimit = data.split(":");
                String IRString = delimit[2] + "" + delimit[3];
                Integer decVal = Integer.parseInt(IRString, 16);
                return (float)decVal;
        }
        public float getVIS() {
                String[] delimit = data.split(":");
                String VisString = delimit[0] + "" + delimit[1];
                Integer decVal = Integer.parseInt(VisString, 16);
                return (float)decVal;
        }
}
```

TEMPERATURE_GRAPHER.java

```
package com.example.android.bluetoothlegatt;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.TimeZone;

import com.jjoe64.graphview.CustomLabelFormatter;
import com.jjoe64.graphview.GraphView;
import com.jjoe64.graphview.GraphView.GraphViewData;
import com.jjoe64.graphview.GraphViewSeries;
import com.jjoe64.graphview.LineGraphView;

import android.app.Activity;
import android.os.Bundle;
import android.os.Environment;
```

```
import android.util.Log;
import android.widget.LinearLayout;

public class TEMPERATURE_GRAPHER extends Activity {

	private final static String TAG = TEMPERATURE_GRAPHER.class.getSimpleName();

	// String file = (String)
	// getIntent().getSerializableExtra("EXTRAS_FILE_NAME");
	// String file = "/sdcard/abc.csv";
	public static String file;

	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.test);

		Bundle extras = TEMPERATURE_GRAPHER.this.getIntent().getExtras();

		ArrayList<GraphViewData> data = new ArrayList<GraphViewData>();

		file = extras.getString("EXTRAS_FILE_NAME");

		file = Environment.getExternalStorageDirectory() + "/" + file;
		// file = "/sdcard/" + file;

		BufferedReader reader = null;
		try {
			reader = new BufferedReader(new FileReader(file));
		} catch (FileNotFoundException e1) {
			// TODO Auto-generated catch block
			e1.printStackTrace();
		}
		try {
			String line;
			while ((line = reader.readLine()) != null) {
				String[] RowData = line.split(",");
				if (!RowData[2].contains("NaN")) {
					String type = RowData[0].replace("", ' ').trim();

					String date = RowData[1].replace("", ' ').trim();

					String skin_temp = RowData[2].replace("", ' ').trim();

					String utc = RowData[3].replace("", ' ').trim();

					if (type.equals("Temperature")) {

						Log.d(TAG, "IN HERE " + type);
						Log.d(TAG, "IN HERE UTC" + (double)
Long.parseLong(utc));
						Log.d(TAG,
								"IN HERE SKINTEMP "
										+
Double.parseDouble(skin_temp));

						data.add(new GraphViewData(
								(double) Long.parseLong(utc), Double

.parseDouble(skin_temp)));
```

```
                                        }
                                }
                        }
                } catch (IOException ex) {
                        // handle exception
                } finally {
                        try {
                                reader.close();
                        } catch (IOException e) {
                                // handle exception
                        }
                }

                GraphView graphView = new LineGraphView(this,
                                "Skin Temperature (Fahrenheit)");

                graphView.setCustomLabelFormatter(new CustomLabelFormatter() {

                        @Override
                        public String formatLabel(double value, boolean isValueX) {
                                if (isValueX) {
                                        return getDateCurrentTimeZone((long) value);
                                }
                                return null;
                        }
                });

                GraphViewData[] input = new GraphViewData[data.size()];

                for (int i = 0; i < input.length; i++) {

                        input[i] = data.get(i);
                }

                // add data
                graphView.addSeries(new GraphViewSeries(input));
                // set view port, start=2, size=40
                graphView.setViewPort(2, 40);
                graphView.setScrollable(true);
                // optional - activate scaling / zooming
                graphView.setScalable(true);

                LinearLayout layout = (LinearLayout) findViewById(R.id.graph1);
                layout.addView(graphView);
        }

        private String getDateCurrentTimeZone(long timestamp) {
                try {
                        Calendar calendar = Calendar.getInstance();
                        TimeZone tz = TimeZone.getDefault();
                        calendar.setTimeInMillis(timestamp * 1000);
                        calendar.add(Calendar.MILLISECOND, 0);
                        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
                        Date currenTimeZone = (Date) calendar.getTime();
                        return sdf.format(currenTimeZone);
                } catch (Exception e) {
                }
                return "";
        }
```

```
        /*
         * int num = 150;
         *
         * double v=0; for (int i=0; i<num; i++) { v += 0.2;
         *
         * }
         */
}
```

TMP006_Obj.java

```
package com.example.android.bluetoothlegatt;

import static java.lang.Math.pow;

import java.io.Serializable;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.TimeZone;


import android.util.Log;

public class TMP006_Obj implements Serializable{

        private String date;
        private String data;

        private byte[] value = new byte[4];


        private double ambient;
        private double target;

        public TMP006_Obj(Integer UTC, String data){

                this.date = getDateCurrentTimeZone(UTC);
                this.data = data;


                Log.d("TEMPERATURE", "HELLO " + data);
                String[] delimit = data.split(":");


                byte vLSB = (byte)((int)(Integer.parseInt(delimit[2],16) & 0xFF));
                byte vMSB = (byte)(((int)(Integer.parseInt(delimit[3],16) & 0xFF)));
                byte tLSB = (byte)((int)(Integer.parseInt(delimit[4],16) & 0xFF));
                byte tMSB = (byte)((int)(Integer.parseInt(delimit[5],16) & 0xFF));

                byte value[] = {vMSB,vLSB,tMSB,tLSB};



                 this.ambient = extractAmbientTemperature(value);
            this.target = extractTargetTemperature(value, ambient);

        }
```

```
        private String getDateCurrentTimeZone(long timestamp) {
    try{
       Calendar calendar = Calendar.getInstance();
       TimeZone tz = TimeZone.getDefault();
       calendar.setTimeInMillis(timestamp * 1000);
       calendar.add(Calendar.MILLISECOND, 0);
       SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
       Date currenTimeZone = (Date) calendar.getTime();
       return sdf.format(currenTimeZone);
    }catch (Exception e) {
    }
    return "";
}

        public double getTemp(){

                return target * 9/5 + 32;
        }



        private static double extractAmbientTemperature(byte [] v) {
            int offset = 2;
            return shortUnsignedAtOffset(v, offset) / 128.0;
          }

          private static double extractTargetTemperature(byte [] v, double ambient) {
            Integer twoByteValue = shortSignedAtOffset(v, 0);

            double Vobj2 = twoByteValue.doubleValue();
            Vobj2 *= 0.00000015625;

            double Tdie = ambient + 273.15;


            double S0 = 5.593E-14; // Calibration factor
            double a1 = 1.75E-3;
            double a2 = -1.678E-5;
            double b0 = -2.94E-5;
            double b1 = -5.7E-7;
            double b2 = 4.63E-9;
            double c2 = 13.4;
            double Tref = 298.15;
            double S = S0 * (1 + a1 * (Tdie - Tref) + a2 * pow((Tdie - Tref), 2));


            double Vos = b0 + b1 * (Tdie - Tref) + b2 * pow((Tdie - Tref), 2);

            double fObj = (Vobj2 - Vos) + c2 * pow((Vobj2 - Vos), 2);



            double data = (Tdie)*(Tdie)*(Tdie)*(Tdie) + (fObj / S);
```

```
                double tObj = pow(data, .25);


                return tObj - 273.15;
            }

            private static Integer shortSignedAtOffset(byte[] c, int offset) {
                    Integer lowerByte = (int) c[offset] & 0xFF;
                    Integer upperByte = (int) c[offset+1]; // // Interpret MSB as signed
                    return (upperByte << 8) + lowerByte;
                }

                private static Integer shortUnsignedAtOffset(byte[] c, int offset) {
                    Integer lowerByte = (int) c[offset] & 0xFF;
                    Integer upperByte = (int) c[offset+1] & 0xFF; // // Interpret MSB as signed
                    return (upperByte << 8) + lowerByte;
                }
}
```

UVLIGHT_GRAPHER.java

```
package com.example.android.bluetoothlegatt;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.TimeZone;

import com.jjoe64.graphview.CustomLabelFormatter;
import com.jjoe64.graphview.GraphView;
import com.jjoe64.graphview.GraphView.GraphViewData;
import com.jjoe64.graphview.GraphViewSeries;
import com.jjoe64.graphview.LineGraphView;

import android.app.Activity;
import android.os.Bundle;
import android.os.Environment;
import android.util.Log;
import android.widget.LinearLayout;

public class UVLIGHT_GRAPHER extends Activity {

        private final static String TAG = UVLIGHT_GRAPHER.class.getSimpleName();

        // String file = (String)
        // getIntent().getSerializableExtra("EXTRAS_FILE_NAME");
        // String file = "/sdcard/abc.csv";
        public static String file;

        protected void onCreate(Bundle savedInstanceState) {
```

```
super.onCreate(savedInstanceState);
setContentView(R.layout.test);

Bundle extras = UVLIGHT_GRAPHER.this.getIntent().getExtras();

ArrayList<GraphViewData> data = new ArrayList<GraphViewData>();

file = extras.getString("EXTRAS_FILE_NAME");

file = Environment.getExternalStorageDirectory() + "/" + file;
// file = "/sdcard/" + file;

BufferedReader reader = null;
try {
	reader = new BufferedReader(new FileReader(file));
} catch (FileNotFoundException e1) {
	// TODO Auto-generated catch block
	e1.printStackTrace();
}
try {
	String line;
	while ((line = reader.readLine()) != null) {
		String[] RowData = line.split(",");
		String type = RowData[0].replace("", ' ').trim();

		String date = RowData[1].replace("", ' ').trim();

		String light = RowData[2].replace("", ' ').trim();

		String utc = RowData[3].replace("", ' ').trim();

		if (type.equals("UV")) {

			Log.d(TAG, "IN HERE " + type);
			Log.d(TAG, "IN HERE UTC" + (double) Long.parseLong(utc));
			Log.d(TAG, "IN HERE UVLIGHT " + Float.parseFloat(light));

			data.add(new GraphViewData((double) Long.parseLong(utc),
					Double.parseDouble(light)));

		}
	}
} catch (IOException ex) {
	// handle exception
} finally {
	try {
		reader.close();
	} catch (IOException e) {
		// handle exception
	}
}

GraphView graphView = new LineGraphView(this, "UV Light (UV Index)");

graphView.setCustomLabelFormatter(new CustomLabelFormatter() {

	@Override
	public String formatLabel(double value, boolean isValueX) {
		if (isValueX) {
			return getDateCurrentTimeZone((long) value);
```

```
                                        }
                                        return null;
                                }
                        });

                        GraphViewData[] input = new GraphViewData[data.size()];

                        for (int i = 0; i < input.length; i++) {

                                input[i] = data.get(i);
                        }

                        // add data
                        graphView.addSeries(new GraphViewSeries(input));
                        // set view port, start=2, size=40
                        graphView.setViewPort(2, 40);
                        graphView.setScrollable(true);
                        // optional - activate scaling / zooming
                        graphView.setScalable(true);

                        LinearLayout layout = (LinearLayout) findViewById(R.id.graph1);
                        layout.addView(graphView);
                }

                private String getDateCurrentTimeZone(long timestamp) {
                        try {
                                Calendar calendar = Calendar.getInstance();
                                TimeZone tz = TimeZone.getDefault();
                                calendar.setTimeInMillis(timestamp * 1000);
                                calendar.add(Calendar.MILLISECOND, 0);
                                SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
                                Date currenTimeZone = (Date) calendar.getTime();
                                return sdf.format(currenTimeZone);
                        } catch (Exception e) {
                        }
                        return "";
                }

                /*
                 * int num = 150;
                 *
                 * double v=0; for (int i=0; i<num; i++) { v += 0.2;
                 *
                 * }
                 */
}
```

VISLIGHT_GRAPHER.java

```
package com.example.android.bluetoothlegatt;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.text.SimpleDateFormat;
```

```
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.TimeZone;

import com.jjoe64.graphview.CustomLabelFormatter;
import com.jjoe64.graphview.GraphView;
import com.jjoe64.graphview.GraphView.GraphViewData;
import com.jjoe64.graphview.GraphViewSeries;
import com.jjoe64.graphview.LineGraphView;

import android.app.Activity;
import android.os.Bundle;
import android.os.Environment;
import android.util.Log;
import android.widget.LinearLayout;

public class VISLIGHT_GRAPHER extends Activity {

	private final static String TAG = VISLIGHT_GRAPHER.class.getSimpleName();

	// String file = (String)
	// getIntent().getSerializableExtra("EXTRAS_FILE_NAME");
	// String file = "/sdcard/abc.csv";
	public static String file;

	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.test);

		Bundle extras = VISLIGHT_GRAPHER.this.getIntent().getExtras();

		ArrayList<GraphViewData> data = new ArrayList<GraphViewData>();

		file = extras.getString("EXTRAS_FILE_NAME");

		file = Environment.getExternalStorageDirectory() + "/" + file;
		// file = "/sdcard/" + file;

		BufferedReader reader = null;
		try {
			reader = new BufferedReader(new FileReader(file));
		} catch (FileNotFoundException e1) {
			// TODO Auto-generated catch block
			e1.printStackTrace();
		}
		try {
			String line;
			while ((line = reader.readLine()) != null) {
				String[] RowData = line.split(",");
				String type = RowData[0].replace("", ' ').trim();

				String date = RowData[1].replace("", ' ').trim();

				String light = RowData[2].replace("", ' ').trim();

				String utc = RowData[3].replace("", ' ').trim();

				if (type.equals("Visible")) {
```

```
                                        Log.d(TAG, "IN HERE " + type);
                                        Log.d(TAG, "IN HERE UTC" + (double) Long.parseLong(utc));
                                        Log.d(TAG, "IN HERE VISLIGHT " + Float.parseFloat(light));

                                        data.add(new GraphViewData((double) Long.parseLong(utc),
                                                        Double.parseDouble(light)));

                                }
                        }
                } catch (IOException ex) {
                        // handle exception
                } finally {
                        try {
                                reader.close();
                        } catch (IOException e) {
                                // handle exception
                        }
                }

                GraphView graphView = new LineGraphView(this, "Visible Light (lux)");

                graphView.setCustomLabelFormatter(new CustomLabelFormatter() {

                        @Override
                        public String formatLabel(double value, boolean isValueX) {
                                if (isValueX) {
                                        return getDateCurrentTimeZone((long) value);
                                }
                                return null;
                        }
                });

                GraphViewData[] input = new GraphViewData[data.size()];

                for (int i = 0; i < input.length; i++) {

                        input[i] = data.get(i);
                }

                // add data
                graphView.addSeries(new GraphViewSeries(input));
                // set view port, start=2, size=40
                graphView.setViewPort(2, 40);
                graphView.setScrollable(true);
                // optional - activate scaling / zooming
                graphView.setScalable(true);

                LinearLayout layout = (LinearLayout) findViewById(R.id.graph1);
                layout.addView(graphView);
        }

        private String getDateCurrentTimeZone(long timestamp) {
                try {
                        Calendar calendar = Calendar.getInstance();
                        TimeZone tz = TimeZone.getDefault();
                        calendar.setTimeInMillis(timestamp * 1000);
                        calendar.add(Calendar.MILLISECOND, 0);
                        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
                        Date currenTimeZone = (Date) calendar.getTime();
                        return sdf.format(currenTimeZone);
```

```
                        } catch (Exception e) {
                        }
                        return "";
                }

                /*
                 * int num = 150;
                 *
                 * double v=0; for (int i=0; i<num; i++) { v += 0.2;
                 *
                 * }
                 */
}
```